BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

THE COMMISSIONER IS AUTHORIZED TO CHARGE ANY DEFICIENCY IN THE FEES FOR THIS PAPER TO DEPOSIT ACCOUNT NO. 23-0975

#3

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of :

Shinichiro ETO et al. :

Serial No. NEW : **Attn: APPLICATION BRANCH**

Filed April 19, 2001 : Attorney Docket No. 2001_0469A

REAL-TIME OS SIMULATOR

CLAIM OF PRIORITY UNDER 35 USC 119

Assistant Commissioner for Patents,
Washington, DC 20231

Sir:

Applicants in the above-entitled application hereby claim the date of priority under the International Convention of Japanese Patent Application No. 2000-120903, filed April 21, 2000, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Shinichiro ETO et al.

By _____
Charles R. Watts
Registration No. 33,142
Attorney for Applicants

CRW/asd
Washington, D.C. 20006-1021
Telephone (202) 721-8200
Facsimile (202) 721-8250
April 19, 2001

日　本　国　特　許　庁
PATENT OFFICE
JAPANESE GOVERNMENT

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２０００年　４月２１日

出願番号
Application Number: 特願２０００－１２０９０３

出願人
Applicant (s): 松下電器産業株式会社

２０００年１２月１５日

特許庁長官
Commissioner,
Patent Office 及川耕造

出証番号　出証特２０００－３１０４１５８

【書類名】　特許願

【整理番号】　2022520081

【あて先】　特許庁長官殿

【国際特許分類】　G06F　9/46

【発明者】

【住所又は居所】　広島県東広島市鏡山３丁目１０番１８号　株式会社松下電器情報システム広島研究所内

【氏名】　衛藤　眞一郎

【発明者】

【住所又は居所】　広島県東広島市鏡山３丁目１０番１８号　株式会社松下電器情報システム広島研究所内

【氏名】　枯木　正吉

【特許出願人】

【識別番号】　000005821

【氏名又は名称】　松下電器産業株式会社

【代理人】

【識別番号】　100098291

【弁理士】

【氏名又は名称】　小笠原　史朗

【手数料の表示】

【予納台帳番号】　035367

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　9405386

【プルーフの要否】　要

【書類名】　　明細書

【発明の名称】　リアルタイムＯＳシミュレータ

【特許請求の範囲】

【請求項１】　リアルタイムＯＳ上で実行される複数のタスクのそれぞれに汎用のマルチスレッドＯＳ上で実行されるタスク処理スレッドを割り当て、前記リアルタイムＯＳの動作を前記マルチスレッドＯＳ上で模倣するリアルタイムＯＳシミュレータであって、

前記リアルタイムＯＳと同一の条件下で前記タスク処理スレッドが発する要求を受け付け、前記要求に応じてタスクの切り替えを指示するタスク切り替え指示手段と、

前記タスク切り替え指示手段と協働して、前記マルチスレッドＯＳの機能を用いて前記タスク処理スレッドを停止および再開させることにより、前記タスク処理スレッドの中から選択した一のタスク処理スレッドを実行させるタスク切り替えスレッドとを備えた、リアルタイムＯＳシミュレータ。

【請求項２】　前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択し、前記タスク切り替えスレッドに対して処理開始を指示した後に、前記要求を発したタスク処理スレッドを停止させ、

前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行されていたタスク処理スレッドが停止した後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする、請求項１に記載のリアルタイムＯＳシミュレータ。

【請求項３】　前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行されていたタスク処理スレッドが停止したか否かを所定の時間間隔で調べることを特徴とする、請求項２に記載のリアルタイムＯＳシミュレータ。

【請求項４】　前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択し、前記タスク切り替えスレッドに対して処理開始を指示した後に、前記要求を発したタスク処理スレッドを待機状態にさせ、

前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行さ

れていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドの待機状態を解除するとともに、当該タスク処理スレッドを再開させることを特徴とする、請求項１に記載のリアルタイムＯＳシミュレータ。

【請求項５】　前記タスク切り替え指示手段は、前記タスク切り替えスレッドが処理可能状態となった後に、前記タスク切り替えスレッドに処理開始を指示することを特徴とする、請求項２または４に記載のリアルタイムＯＳシミュレータ。

【請求項６】　前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択した後に、前記タスク切り替えスレッドに対して処理開始を指示し、

前記タスク切り替えスレッドは、前記タスク処理スレッドよりも高い優先度で実行され、処理開始を指示された時には、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする、請求項１に記載のリアルタイムＯＳシミュレータ。

【請求項７】　前記タスク処理スレッドを生成するタスク処理スレッド生成手段をさらに備えた、請求項１に記載のリアルタイムＯＳシミュレータ。

【請求項８】　擬似的に割り込みを発生させる割り込みスレッドが発する割り込み要求を受け付け、実行中のタスク処理スレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、次に実行すべきタスク処理スレッドを選択して再開させる割り込み処理手段をさらに備えた、請求項１に記載のリアルタイムＯＳシミュレータ。

【請求項９】　前記割り込み処理手段は、前記割り込みスレッドから割り込み要求を受け付けた時に他の割り込みスレッドが実行されていた場合には、実行中の割り込みスレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、停止させた割り込みスレッドを再開させることを特徴とする、請求項８に記載のリアルタイムＯＳシミュレータ。

【請求項１０】　前記割り込みスレッドの中に、所定の時間間隔で擬似的に割り込みを発生させるシステムクロック割り込みスレッドが含まれることを特徴とする、請求項８に記載のリアルタイムＯＳシミュレータ。

【請求項１１】　前記割り込みスレッドを生成する割り込みスレッド生成手段をさらに備えた、請求項８に記載のリアルタイムＯＳシミュレータ。

【請求項１２】　リアルタイムＯＳ上で実行される複数のタスクのそれぞれに汎用のマルチスレッドＯＳ上で実行されるタスク処理スレッドを割り当て、前記リアルタイムＯＳの動作を前記マルチスレッドＯＳ上で模倣するリアルタイムＯＳのシミュレーション方法をコンピュータで実行させるための記録媒体を記録したコンピュータ読み取り可能な記録媒体であって、

前記リアルタイムＯＳと同一の条件下で前記タスク処理スレッドが発する要求を受け付け、前記要求に応じてタスクの切り替えを指示する第１のステップと、

前記マルチスレッドＯＳの機能を用いて前記タスク処理スレッドを停止および再開させることにより、前記タスク処理スレッドの中から選択した一のタスク処理スレッドを実行させる第２のステップとを備えた、リアルタイムＯＳのシミュレーション方法をコンピュータに実行させるためのプログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１３】　前記第１のステップは、次に実行すべきタスク処理スレッドを選択した後に、前記要求を発したタスク処理スレッドを停止させ、

前記第２のステップは、直前に実行されていたタスク処理スレッドが停止した後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする、請求項１２に記載の記録媒体。

【請求項１４】　前記第１のステップは、次に実行すべきタスク処理スレッドを選択した後に、前記要求を発したタスク処理スレッドを待機状態にさせ、

前記第２のステップは、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドの待機状態を解除するとともに、当該タスク処理スレッドを再開させることを特徴とする、請求項１２に記載の記録媒体。

【請求項１５】　前記第１のステップは、次に実行すべきタスク処理スレッドを選択し、

前記第２のステップは、前記タスク処理スレッドよりも高い優先度で実行され、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく

選択されたタスク処理スレッドを再開させることを特徴とする、請求項１２に記載の記録媒体。

【請求項１６】　前記リアルタイムＯＳのシミュレーション方法は、擬似的に割り込みを発生させる割り込みスレッドが発する割り込み要求を受け付け、実行中のタスク処理スレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、次に実行すべきタスク処理スレッドを選択して再開させる第３のステップをさらに備えた、請求項１２に記載の記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、組み込み機器に内蔵されるアプリケーションソフトウェアを開発するために使用されるリアルタイムＯＳシミュレータに関する。

【０００２】

【従来の技術】

携帯電話やデジタルＴＶなどの組み込み機器は、機器ごとに固有のアプリケーションソフトウェアを内蔵している。組み込み機器用のソフトウェアは、リアルタイムＯＳ上で実行されるマルチタスクのソフトウェアとして実現され、ブレッドボードやＩＣＥ（Ｉｎ　Ｃｉｒｃｕｉｔ　Ｅｍｕｌａｔｏｒ）などからなる開発環境（以下、実システムという）を用いて開発される。しかし、実システムを用いる方法では、実システム完成後のソフトウェア開発期間が短くなり、実システムがソフトウェア開発者に比べて不足するといった問題が生じる。また、実システムの動作周波数上昇に伴い、ＩＣＥを用いたデバッグ自体が困難になるという問題も生じている。

【０００３】

この問題を解決する１つの手法として、汎用のパソコンやワークステーション上に、実システムと同様の動作を行うソフトウェア開発システムを構成し、これを用いて組み込み機器用のソフトウェアを開発する方法が考えられる。例えば、実システムにおけるタスクを汎用のマルチスレッドＯＳ（以下、汎用ＯＳという）上のスレッドと１対１に対応づけ、リアルタイムＯＳの動作を模倣する「リア

ルタイムＯＳシミュレータ」を用いる方法がある。この方法によれば、汎用ＯＳが提供するマルチスレッド機能を用いてリアルタイムＯＳのマルチタスク機能を模倣すればよいので、リアルタイムＯＳシミュレータを容易に実現することができる。また、リアルタイムＯＳシミュレータのみを開発すれば、十分な数のソフトウェア開発システムをソフトウェア開発者に提供し、ソフトウェアの開発効率と信頼性とを向上させることができる。なお、以下でスレッドという時には、一部の汎用ＯＳにおいてプロセスと呼ばれるものを含むものとする。

【０００４】

リアルタイムＯＳは、タスク同期管理、タイマー管理、タスク間通信、および、メモリ管理などの様々な機能を有する。リアルタイムＯＳは、このうち、実行中のタスクを切り替えるディスパッチ処理、および、発生した割り込みに対して所定の割り込みハンドラを呼び出す割り込み処理を実現するにあたり、実システムのハードウェアを用いる。これに対し、リアルタイムＯＳシミュレータは、実システムのハードウェアを用いることができないため、この２つの処理を行うことが難しい。

【０００５】

リアルタイムＯＳは、割り込みハンドラを実行中に他の割り込みが発生した場合に、後に発生した割り込みを先に処理する多重割り込み処理を行う場合がある。また、リアルタイムＯＳは、タスクまたは割り込みハンドラ内で呼び出された時に、必要に応じてディスパッチ処理を行う。タスク内でリアルタイムＯＳが呼び出された時には、その時点でディスパッチ処理が行われる。これに対し、割り込みハンドラ内でリアルタイムＯＳが呼び出された時には、割り込みハンドラが終了した時点でディスパッチ処理（以下、遅延ディスパッチ処理という）が行われる。リアルタイムＯＳシミュレータは、このような遅延ディスパッチ処理や多重割り込み処理を含めて、リアルタイムＯＳの機能を汎用ＯＳ上で忠実に模倣する必要がある。

【０００６】

図１８は、日本国特許第２８２０１８９号によって開示された、数値制御装置の制御ソフトウェア実行システムのソフトウェア構成図である。なお、図１８は

、本願発明との対比を容易にするために、発明の本質を損なわない程度に抽象化されている。図１８に示す４つのスレッドは、汎用ＯＳを搭載したホスト計算機上で並行に実行される。３つのスレッド９１～９３は、このホスト計算機と異なる数値制御装置において実行されるタスクと同じ処理を行う。スケジューラスレッド９０は、スレッド９１～９３に対して定期的に割り込み、スレッドの実行を能動的に切り替える。このソフトウェア実行システムによれば、数値制御装置の制御ソフトウェアをホスト計算機上で動作させることができる。

【０００７】

【発明が解決しようとする課題】

しかしながら、図１８に示すスケジューラスレッド９０は、以下に示すように、組み込み機器用のソフトウェア開発において使用されるリアルタイムＯＳシミュレータに必要な機能を果たすことができない。

【０００８】

まず、スケジューラスレッド９０は、スレッド９１～９３に対して定期的に割り込み、実行スレッドを能動的に切り替えるプリエンプティブなスレッド制御を行う。これに対して、実システムでは、タスクがリアルタイムＯＳを呼び出した時に実行タスクを切り替えるノンプリエンプティブなタスク制御が行われる。従来の方法では、リアルタイムＯＳの特徴であるノンプリエンプティブなタスク制御を行うことができない。

【０００９】

次に、スケジューラスレッド９０は、汎用ＯＳがすべてのスレッドを平等に制御するとの仮定のもとで、スレッド９１～９３に定期的に割り込み、実行スレッドを切り替える。しかし、一般に汎用ＯＳのスレッドスケジューリングアルゴリズムは公開されない。また、一部の汎用ＯＳは、指定されたスレッド優先度に基づき内部で別途スレッド優先度を算出し、算出したスレッド優先度に従ってスレッドの実行を制御する。このような汎用ＯＳではアプリケーションソフトウェアでスレッド優先度を設定するだけでは、スレッドの実行を制御することができない。したがって、このような汎用ＯＳを用いて従来の方法を実施する場合には、スレッド９１～９３のみが実行され、スケジューラスレッド９０が全く実行され

ない可能性がある。

【００１０】

さらに、スケジューラスレッド９０は、リアルタイムＯＳシミュレータに必要とされる割り込み処理機能を有していない。組み込み機器用のソフトウェアは、外部からの割り込みに応答して動作する。しかし、従来の方法では、このような割り込み処理を扱うことができない。

【００１１】

それ故に、本発明は、任意のスレッドスケジューリングアルゴリズムを用いた汎用のマルチスレッドＯＳ上においても、リアルタイムＯＳのディスパッチ処理と割り込み処理とを忠実に模倣するリアルタイムＯＳシミュレータを提供することを目的とする。

【００１２】

【課題を解決するための手段および発明の効果】

第１の発明は、リアルタイムＯＳ上で実行される複数のタスクのそれぞれに汎用のマルチスレッドＯＳ上で実行されるタスク処理スレッドを割り当て、前記リアルタイムＯＳの動作を前記マルチスレッドＯＳ上で模倣するリアルタイムＯＳシミュレータであって、

前記リアルタイムＯＳと同一の条件下で前記タスク処理スレッドが発する要求を受け付け、前記要求に応じてタスクの切り替えを指示するタスク切り替え指示手段と、

前記タスク切り替え指示手段と協働して、前記マルチスレッドＯＳの機能を用いて前記タスク処理スレッドを停止および再開させることにより、前記タスク処理スレッドの中から選択した一のタスク処理スレッドを実行させるタスク切り替えスレッドとを備える。

【００１３】

このような第１の発明によれば、タスク切り替え指示手段とタスク切り替えスレッドとの作用により、一のタスク処理スレッドのみが切り替えて実行される。これにより、マルチスレッドＯＳが提供するスレッドスケジューリングアルゴリズムにかかわらず、リアルタイムＯＳのディスパッチ処理を模倣することができ

る。

【００１４】

第２の発明は、第１の発明において、前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択し、前記タスク切り替えスレッドに対して処理開始を指示した後に、前記要求を発したタスク処理スレッドを停止させ、

前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行されていたタスク処理スレッドが停止した後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする。

【００１５】

このような第２の発明によれば、先に実行されていたタスク処理スレッドが停止した後に、タスク切り替えスレッドによって次に実行すべきタスク処理スレッドが再開される。このため、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【００１６】

第３の発明は、第２の発明において、前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行されていたタスク処理スレッドが停止したか否かを所定の時間間隔で調べることを特徴とする。

【００１７】

このような第３の発明によれば、マルチスレッドＯＳが提供するスレッドスケジューリングアルゴリズムにかかわらず、先に実行されていたタスク処理スレッドが停止した後に、タスク切り替えスレッドを実行させることができる。

【００１８】

第４の発明は、第１の発明において、前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択し、前記タスク切り替えスレッドに対して処理開始を指示した後に、前記要求を発したタスク処理スレッドを待機状態にさせ、

前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドの待機状態を解除するとともに、当該タスク処理スレッドを再開

させることを特徴とする。

【００１９】

このような第４の発明によれば、タスク切り替えスレッドが先に実行されていたタスク処理スレッドを停止させた後に、次に実行すべきタスク処理スレッドが再開される。このため、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【００２０】

第５の発明は、第２または第４の発明において、前記タスク切り替え指示手段は、前記タスク切り替えスレッドが処理可能状態となった後に、前記タスク切り替えスレッドに処理開始を指示することを特徴とする。

【００２１】

このような第５の発明によれば、複数のタスク処理スレッドが同時にタスク切り替えスレッドに処理開始を指示することがない。このため、タスク切り替えスレッドは、ディスパッチ処理を確実に行うことができる。

【００２２】

第６の発明は、第１の発明において、前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択した後に、前記タスク切り替えスレッドに対して処理開始を指示し、

前記タスク切り替えスレッドは、前記タスク処理スレッドよりも高い優先度で実行され、処理開始を指示された時には、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする。

【００２３】

このような第６の発明によれば、指定された優先度どおりにスレッドを実行させるマルチスレッドＯＳを用いた場合に、タスク切り替えスレッドが、タスク処理スレッドよりも優先的に実行される。このため、タスク処理スレッドに待機状態を設けなくても、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【００２４】

第７の発明は、第１の発明において、前記タスク処理スレッドを生成するタス

ク処理スレッド生成手段をさらに備える。

【００２５】

このような第７の発明によれば、タスク処理スレッドをリアルタイムＯＳシミュレータによって生成し、リアルタイムＯＳのディスパッチ処理を模倣することができる。

【００２６】

第８の発明は、第１の発明において、擬似的に割り込みを発生させる割り込みスレッドが発する割り込み要求を受け付け、実行中のタスク処理スレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、次に実行すべきタスク処理スレッドを選択して再開させる割り込み処理手段をさらに備える。

【００２７】

このような第８の発明によれば、割り込みスレッドは、割り込み処理手段を用いることにより、実行中のタスク処理スレッドを停止させて、割り込みハンドラを呼び出す。これにより、マルチスレッドＯＳが提供するスレッドスケジューリングアルゴリズムにかかわらず、リアルタイムＯＳの割り込み処理と遅延ディスパッチ処理とを模倣することができる。

【００２８】

第９の発明は、第８の発明において、前記割り込み処理手段は、前記割り込みスレッドから割り込み要求を受け付けた時に他の割り込みスレッドが実行されていた場合には、実行中の割り込みスレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、停止させた割り込みスレッドを再開させることを特徴とする。

【００２９】

このような第９の発明によれば、割り込みスレッドは、割り込み処理手段を用いることにより、実行中の割り込みスレッドを停止させて、後の割り込みに対応した割り込みハンドラを先に呼び出す。これにより、リアルタイムＯＳの多重割り込み処理を模倣することができる。

【００３０】

第１０の発明は、第８の発明において、前記割り込みスレッドの中に、所定の時間間隔で擬似的に割り込みを発生させるシステムクロック割り込みスレッドが含まれることを特徴とする。

【００３１】

このような第１０の発明によれば、システムクロック割り込みスレッドによって所定の時間間隔で割り込みを擬似的に発生させることにより、リアルタイムＯＳのタイマー管理機能を模倣することができる。

【００３２】

第１１の発明は、第８の発明において、前記割り込みスレッドを生成する割り込みスレッド生成手段をさらに備える。

【００３３】

このような第１１の発明によれば、割り込みスレッドをリアルタイムＯＳシミュレータによって生成し、リアルタイムＯＳの割り込み処理を模倣することができる。

【００３４】

第１２の発明は、リアルタイムＯＳ上で実行される複数のタスクのそれぞれに汎用のマルチスレッドＯＳ上で実行されるタスク処理スレッドを割り当て、前記リアルタイムＯＳの動作を前記マルチスレッドＯＳ上で模倣するリアルタイムＯＳのシミュレーション方法をコンピュータで実行させるための記録媒体を記録したコンピュータ読み取り可能な記録媒体であって、

前記リアルタイムＯＳと同一の条件下で前記タスク処理スレッドが発する要求を受け付け、前記要求に応じてタスクの切り替えを指示する第１のステップと、

前記マルチスレッドＯＳの機能を用いて前記タスク処理スレッドを停止および再開させることにより、前記タスク処理スレッドの中から選択した一のタスク処理スレッドを実行させる第２のステップとを備える。

【００３５】

このような第１２の発明によれば、一のタスク処理スレッドのみが切り替えて実行されるので、マルチスレッドＯＳが提供するスレッドスケジューリングアルゴリズムにかかわらず、リアルタイムＯＳのディスパッチ処理を模倣することが

できる。

【００３６】

第１３の発明は、第１２の発明において、前記第１のステップは、次に実行すべきタスク処理スレッドを選択した後に、前記要求を発したタスク処理スレッドを停止させ、

前記第２のステップは、直前に実行されていたタスク処理スレッドが停止した後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする。

【００３７】

このような第１３の発明によれば、先に実行されていたタスク処理スレッドが停止した後に、第２のステップより次に実行すべきタスク処理スレッドが再開される。このため、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【００３８】

第１４の発明は、第１２の発明において、前記第１のステップは、次に実行すべきタスク処理スレッドを選択した後に、前記要求を発したタスク処理スレッドを待機状態にさせ、

前記第２のステップは、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドの待機状態を解除するとともに、当該タスク処理スレッドを再開させることを特徴とする。

【００３９】

このような第１４の発明によれば、第２のステップにより、先に実行されていたタスク処理スレッドが停止された後に、次に実行すべきタスク処理スレッドが再開される。このため、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【００４０】

第１５の発明は、第１２の発明において、前記第１のステップは、次に実行すべきタスク処理スレッドを選択し、

前記第２のステップは、前記タスク処理スレッドよりも高い優先度で実行され、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく

選択されたタスク処理スレッドを再開させることを特徴とする。

【００４１】

このような第１５の発明によれば、指定された優先度どおりにスレッドを実行させるマルチスレッドＯＳを用いた場合に、第２のステップが、タスク処理スレッドよりも優先的に実行される。このため、タスク処理スレッドに待機状態を設けなくても、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【００４２】

第１６の発明は、第１２の発明において、前記リアルタイムＯＳのシミュレーション方法は、擬似的に割り込みを発生させる割り込みスレッドが発する割り込み要求を受け付け、実行中のタスク処理スレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、次に実行すべきタスク処理スレッドを選択して再開させる第３のステップをさらに備える。

【００４３】

このような第１６の発明によれば、第３のステップにより、実行中のタスク処理スレッドが停止され、割り込みハンドラが呼び出される。これにより、マルチスレッドＯＳが提供するスレッドスケジューリングアルゴリズムにかかわらず、リアルタイムＯＳの割り込み処理を模倣することができる。

【００４４】

【発明の実施の形態】

（第１の実施形態）

図１は、本発明の第１の実施形態に係るリアルタイムＯＳシミュレータが動作するコンピュータシステムのハードウェア構成図である。図１に示すコンピュータ１は、ＣＰＵ２、タイマー回路３、メインメモリ４、ハードディスク５、キーボード６、マウス７、ディスプレイ８および通信ポート９を備える。ＣＰＵ２は、ハードディスク５に予め記憶されたソフトウェアをメインメモリ４に転送して実行する。タイマー回路３は、所定の時間間隔で割り込み信号を発生させ、ＣＰＵ２に対して出力する。コンピュータ１は、キーボード６やマウス７から入力される利用者からの指示に従い、ディスプレイ８に対して画面表示を行う。また、コンピュータ１は、通信ポート９を経由して他のコンピュータなどとデータ通信

を行う。コンピュータ１には、汎用のマルチスレッドＯＳが搭載される。汎用ＯＳ上で実行される各スレッドは、自スレッドまたは他のスレッドを任意のタイミングで停止または再開させることができる。また、この汎用ＯＳは、イベント機能を有するものとする。イベント機能によれば、各スレッドは、特定のイベントを発生させ、あるいは、特定のイベントが発生するまで待機することができる。

【００４５】

図２は、本発明の第１の実施形態に係るリアルタイムＯＳシミュレータのソフトウェア構成図である。実システムでは、複数のタスクがリアルタイムＯＳ上で並行に実行され、複数の割り込みが非同期に発生する。リアルタイムＯＳシミュレータ１０では、実システムにおける各タスクは、汎用ＯＳ上で実行されるスレッド（以下、タスク処理スレッドという）にそれぞれ対応づけられる。実システムにおける各割り込みも、同様に汎用ＯＳ上で実行されるスレッド（以下、割り込みスレッドという）にそれぞれ対応づけられる。

【００４６】

図２に示したリアルタイムＯＳシミュレータ１０では、例として、３つのタスクと３つの割り込みとに対応して、第１から第３のタスク処理スレッド２１～２３と、第１から第３の割り込みスレッド３１～３３とが並行に実行される。各タスク処理スレッド２１～２３は、それぞれ、実システムにおけるタスクの機能を行うタスク関数を呼び出す。また、タスク処理スレッドは、コンピュータ１の記憶装置や入出力機器に対して入出力を行うデバッグ用の入出力関数を呼び出す場合がある。例えば、タスク処理スレッドは、入出力関数を呼び出ことにより、汎用ＯＳの機能を用いてハードディスク５からデータを読み出し、ディスプレイ８にデータを表示する。

【００４７】

割り込みスレッド３１～３３は、実システムにける割り込みを模倣して、擬似的な割り込みを発生させる。また、各割り込みスレッドは、各割り込みに対応した割り込みハンドラを含む。擬似的な割り込みを発生させるタイミングは、コンピュータ１の入力機器などにより与えられ、汎用ＯＳの機能によって各割り込みスレッドに供給される。例えば、タイマー回路３によるタイマー割り込み、キー

ボード６やマウス７などにおける利用者の操作、あるいは、通信ポート９におけるデータ送受信などが、擬似的な割り込みを発生させるタイミングとなる。本実施形態では、割り込みスレッド３３は、システムクロック割り込みに対応して擬似的な割り込みを発生させるシステムクロック割り込みスレッドであるとする。

【００４８】

リアルタイムＯＳシミュレータ１０は、チェンジャースレッド１１、システム関数１２、割り込み処理関数１３、および、システムクロック割り込みスレッド３３を備える。システム関数１２は、実システムにおいてタスクがリアルタイムＯＳを呼び出す場合と同じ方法で、各タスク処理スレッド２１～２３から呼び出される。また、割り込み処理関数１３は、割り込みスレッド３１～３３が擬似的な割り込みを発生させた時に呼び出される。チェンジャースレッド１１は、タスク処理スレッド２１～２３および割り込みスレッド３１～３３と並行に実行される。チェンジャースレッド１１、システム関数１２、および、割り込み処理関数１３からは、後述するように、汎用ＯＳが提供するスレッド停止／再開機能とイベント機能とが呼び出され、これによりスレッドの実行が制御される。

【００４９】

リアルタイムＯＳシミュレータ１０は、タスク関数および割り込みハンドラとともに、メインメモリ４上に転送され、ＣＰＵ２によって実行される。これにより、実システムにおけるタスクに対応したタスク処理スレッド２１～２３は、ハードディスク５やディスプレイ８に対してデータの入出力を行うことができる。また、タスク処理スレッドが実行される間に、擬似的な割り込みを発生させることができる。このように、利用者は、図１に示すコンピュータ１上で図２に示すソフトウェアを実行することにより、実システムにおけるアプリケーションソフトウェアに含まれるタスク関数と割り込みハンドラとをデバッグすることができる。

【００５０】

以下、リアルタイムＯＳシミュレータ１０を構成するソフトウェアの詳細を説明する。図３は、リアルタイムＯＳシミュレータ１０のメイン処理のフローチャートである。リアルタイムＯＳシミュレータ１０には、実システムにおけるタス

クの機能を行うタスク関数と、実システムにおける割り込みハンドラとが提供される。リアルタイムＯＳシミュレータ１０は、まず、チェンジャースレッド１１とシステムクロック割り込みスレッド３３とを生成する（ステップＳ１０１、Ｓ１０２）。次に、リアルタイムＯＳシミュレータ１０は、与えられたタスク関数のそれぞれに対応して、タスク処理スレッドを順次生成する（ステップＳ１０３、Ｓ１０４）。その後、リアルタイムＯＳシミュレータ１０は、割り込みスレッドからの割り込みを許可し（ステップＳ１０５）、以降は所定の時間だけスリープする処理を繰り返す（ステップＳ１０６）。リアルタイムＯＳシミュレータ１０がステップＳ１０６に到達した時点から、図２に示す７つのスレッドが並行に実行される。以降のリアルタイムＯＳシミュレータ１０の機能は、チェンジャースレッド１１、システム関数１２、および、割り込み処理関数１３によって実現される。

【００５１】

リアルタイムＯＳシミュレータ１０は、概ね以下のように動作する。タスク処理スレッド２１～２３は、システム関数１２を呼び出した時にタスクの切り替えを行う場合には、次に実行すべきタスク処理スレッドを選択して、チェンジャースレッド１１に対してディスパッチ処理開始を指示した後に、自ら停止する。チェンジャースレッド１１は、タスク処理スレッドから処理開始を指示されると、次に実行すべく選択されたタスク処理スレッドを再開させる。また、割り込みスレッド３１～３３は、他のスレッドと並行に実行され、擬似的な割り込みを発生させるタイミングで割り込み処理関数１３を呼び出す。割り込み処理関数１３を呼び出した割り込みスレッドは、実行中のスレッドを停止させ、自スレッドを実行中のスレッドとして記録する。その後、割り込みスレッドは、対応した割り込みハンドラを呼び出した後に、次に実行すべきスレッドを選択して再開させる。スレッド間の同期を実現するため、２種類のイベントが用いられる。第１のイベントは、タスク処理スレッド２１～２３がチェンジャースレッド１１に対してディスパッチ処理開始を指示するイベントである。第２のイベントは、チェンジャースレッド１１と割り込みスレッドとが他のスレッドに対してディスパッチ処理や割り込み処理が開始可能となった旨を通知するイベントである。

【００５２】

図４は、チェンジャースレッド１１の動作を示すフローチャートである。チェンジャースレッド１１は、最初に第２のイベントを設定した後（ステップＳ２０１）、ステップＳ２０２からＳ２０７までの処理を繰り返す。チェンジャースレッド１１は、まず、ディスパッチ処理開始を指示する第１のイベント待ち状態となり（ステップＳ２０２）、第１のイベントが設定された後にステップＳ２０３に進む。次に、チェンジャースレッド１１は、実行中のタスク処理スレッドが停止するまで、所定の時間だけスリープする処理を繰り返す（ステップＳ２０３、Ｓ２０４）。これにより、タスク処理スレッドが実行されている間、チェンジャースレッド１１は実行されないことが保証される。実行中のタスク処理スレッドが停止した後、チェンジャースレッド１１は、次に実行すべく選択されたタスク処理スレッドを実行スレッドとして記録し（ステップＳ２０５）、そのタスク処理スレッドを再開させる（ステップＳ２０６）。その後、チェンジャースレッド１１は、ディスパッチ処理や割り込み処理が開始可能となった旨を示す第２のイベントを設定し（ステップＳ２０７）、ステップＳ２０２に戻る。

【００５３】

図５は、タスク処理スレッドから呼び出されるシステム関数１２のフローチャートである。システム関数を呼び出したタスク処理スレッドは、まず、リアルタイムＯＳが提供する機能のうち、スレッドのディスパッチ以外の処理を行う（ステップＳ３０１）。例えば、リアルタイムＯＳにおける実行タスク切り替え処理では、実行中のタスクがタスク待ち行列の最後尾に移動され、タスク待ち行列の先頭にあるタスクが実行されるようになる。これに対応して、リアルタイムＯＳシミュレータ１０でも、ステップＳ３０１において同様の処理が行われる。次に、タスク処理スレッドは、ステップＳ３０１における処理の結果、ディスパッチ処理を行うか否かを判断し（ステップＳ３０２）、行う場合にはステップＳ３０３からＳ３０５までの処理を行う。

【００５４】

タスク処理スレッドは、ディスパッチ処理を行う場合には、まず、第２のイベント待ち状態となり、チェンジャースレッド１１がディスパッチ処理を開始可能

となるまで待機する（ステップＳ３０３）。チェンジャースレッド１１がディスパッチ処理を開始可能となった後、タスク処理スレッドは、チェンジャースレッド１１に対してディスパッチ処理開始を指示する第１のイベントを設定し（ステップＳ３０４）、自スレッドを停止させる（ステップＳ３０５）。これにより、ステップＳ３０５以降、タスク処理スレッドは停止し、チェンジャースレッド１１が実行される。

【００５５】

図６ないし図８は、本実施形態に係るリアルタイムＯＳシミュレータによるディスパッチ処理のタイミングチャートの例である。以下に示すタイミングチャートでは、水平方向の直線はそれぞれ１つのスレッドを表し、直線の種類はスレッドの状態を表す。太線、細線および破線は、それぞれ、スレッドが実行状態、実行可能状態、および、停止状態または待機状態にあることを示す。また、●印はシステム関数の呼び出しを、黒塗り三角印はイベント設定を、△印はイベント待ちを、□印はスレッドの停止を、黒塗り四角印はスレッドの再開を、◆印は実行スレッド記録処理をそれぞれ表す。

【００５６】

図６は、実システムにおける第１のタスクから第２のタスクへの切り替えに対応した、ディスパッチ処理のタイミングチャートの例である。初期状態では、チェンジャースレッドは第１のイベント待ち状態にあり（ステップＳ２０２）、第１のタスク処理スレッドが実行されているとする。第１のタスク処理スレッドは、時刻ｔ１でシステム関数を呼び出し、時刻ｔ１から時刻ｔ２の間にディスパッチ以外の処理を行う（ステップＳ３０１）。次に、第１のタスク処理スレッドは、ディスパッチ処理を行うと判断し（ステップＳ３０２）、時刻ｔ２で第２のイベント待ち状態となる（ステップＳ３０３）。この時点で第２のイベントは既に設定されているので、第１のタスク処理スレッドは、直ちにステップＳ３０４に進み、時刻ｔ３で第１のイベントを設定する。このため、時刻ｔ３以降、チェンジャースレッドは実行可能状態となる。

【００５７】

図６に示すタイミングチャートでは、第１のタスク処理スレッドが引き続き実

行され、時刻ｔ４で自スレッドを停止させる（ステップＳ３０５）。実行中のタスク処理スレッドが時刻ｔ４で停止した後、チェンジャースレッドは、時刻ｔ５で第２のタスク処理スレッドを実行スレッドとして記録し（ステップＳ２０５）、時刻ｔ６で第２のタスク処理スレッドを再開させる（ステップＳ２０６）。このため、時刻ｔ６以降、第２のタスク処理スレッドは、実行可能状態となる。チェンジャースレッドは、時刻ｔ７で第２のイベントを設定し、ディスパッチ処理を開始可能となった旨を通知する（ステップＳ２０７）。その後、チェンジャースレッド１１は、時刻ｔ８で第１のイベント待ち状態となる（ステップＳ２０２）。このため、時刻ｔ８以降、第２のタスク処理スレッドが実行される。このようにして、第１のタスク処理スレッドから第２のタスク処理スレッドへのディスパッチ処理が行われる。ディスパッチに必要な時間は、図６において期間Ｔで示した時刻ｔ２からｔ７までの時間である。

【００５８】

図６に示すタイミングチャートでは、時刻ｔ３から時刻ｔ４までの期間、および、時刻ｔ６から時刻ｔ８までの期間で、２つのスレッドが同時に実行可能状態となる。このような場合にいずれのスレッドが実行されるかは、汎用ＯＳが提供するスレッドスケジューリングアルゴリズムに依存する。

【００５９】

図７は、時刻ｔ３以降にチェンジャースレッドが実行される場合のタイミングチャートの例である。この場合、チェンジャースレッドは、実行中のタスク処理スレッドが停止するまで、所定の時間だけスリープする処理を繰り返す（ステップＳ２０３、Ｓ２０４）。チェンジャースレッドは時刻ｔ３－１でスリープ状態に入り、これ以降、第１のタスク処理スレッドが再び実行される。第１のタスク処理スレッドは、時刻ｔ４で自スレッドを停止させる（ステップＳ３０５）。チェンジャースレッドは、時刻ｔ４－１でスリープ状態から抜け、第１のタスク処理スレッドが停止した後に、ステップＳ２０５からＳ２０７までの処理を行う。このため、時刻ｔ４－１以降のタイミングチャートは、図６と一致する。このようにチェンジャースレッドは実行中のタスク処理スレッドが停止するまでスリープする処理を繰り返すので、汎用ＯＳが提供するスレッドスケジューリングアル

ゴリズムにかかわらず、リアルタイムＯＳのディスパッチ処理を模倣することができる。

【００６０】

図８は、時刻ｔ６以降に第２のタスク処理スレッドが実行される場合のタイミングチャートの例である。初期状態から時刻ｔ６までのタイミングチャートは、図６と一致する。時刻ｔ６で再開された第２のタスク処理スレッドは、時刻ｔ６－１でシステム関数を呼び出し、時刻ｔ６－２で第２のイベント待ち状態となる（ステップＳ３０３）。このため、時刻ｔ６－２以降、チェンジャースレッドが再び実行され、時刻ｔ７で第２のイベントを設定する（ステップＳ２０７）。このように汎用ＯＳが提供するイベント機能を用いることにより、複数のタスク処理スレッドが同時にディスパッチ処理を開始することがない。このため、チェンジャースレッドは、確実にディスパッチ処理を行うことができる。

【００６１】

図９は、割り込みスレッドから呼び出される割り込み処理関数１３のフローチャートである。割り込み処理関数を呼び出した割り込みスレッドは、まず、第２のイベント待ち状態となり、チェンジャースレッドが割り込み処理を開始可能となるまで待機する（ステップＳ４０１）。チェンジャースレッドが割り込み処理を開始可能となった後、割り込みスレッドは、実行中のスレッドを停止させる（ステップＳ４０２）。ステップＳ４０２において停止されるスレッドは、タスク処理スレッドまたは他の割り込みスレッドのいずれかである。次に、割り込みスレッドは、自スレッドを実行スレッドとして記録し（ステップＳ４０３）、第２のイベントを設定して割り込み処理を開始可能となった旨を通知した後に（ステップＳ４０４）、自らの割り込みに対応した割り込みハンドラを呼び出す（ステップＳ４０５）。

【００６２】

割り込みハンドラの処理が終了した後、割り込みスレッドは、多重割り込み処理と遅延ディスパッチ処理とを行うか否かを調べる（ステップＳ４０６、Ｓ４０７）。多重割り込み処理を行わずに遅延ディスパッチ処理を行う場合には、割り込みスレッドは、次に実行すべきタスク処理スレッドを実行スレッドとして記録

し（ステップＳ４０８）、そのスレッドを再開させる（ステップＳ４０９）。それ以外の場合には、割り込みスレッドは、ステップＳ４０２において停止させたスレッドを実行スレッドとして記録し（ステップＳ４１０）、そのスレッドを再開させる（ステップＳ４１１）。

【００６３】

図１０ないし図１３は、本実施形態に係るリアルタイムＯＳシミュレータによる割り込み処理のタイミングチャートの例である。これらの図は、図６と同じ記法を用いて記述されている。

【００６４】

図１０は、第１のタスク処理スレッドが実行されている間に第１の割り込みが発生した場合のタイミングチャートである。初期状態では、第２のイベントは設定され、第１のタスク処理スレッドが実行されているとする。第１の割り込みスレッドは、初期状態では実行可能状態にあり、時刻ｔ１で割り込み処理関数を呼び出すとする。時刻ｔ１以降、第１の割り込みスレッドは、図９に示したフローチャートに従って動作する。第１の割り込みスレッドは、まず、時刻ｔ２で第２のイベント待ち状態となる（ステップＳ４０１）。この時点で第２のイベントは既に設定されているので、第１の割り込みスレッドは、直ちにステップＳ４０２に進み、時刻ｔ３で直前に実行されていたスレッド、すなわち、第１のタスク処理スレッドを停止させる（ステップＳ４０２）。このため、時刻ｔ３以降、第１のタスク処理スレッドは、停止状態となる。

【００６５】

次に、第１の割り込みスレッドは、時刻ｔ４で自スレッドを実行スレッドとして記録し（ステップＳ４０３）、時刻ｔ５で第２のイベントを設定する（ステップＳ４０４）。このため、時刻ｔ５以降、他の割り込み処理が開始可能となる。さらに、第１の割り込みスレッドは、時刻ｔ６から時刻ｔ７までの期間で自らの割り込みに対応した割り込みハンドラを呼び出す（ステップＳ４０５）。その後、第１の割り込みスレッドは、多重割り込み処理および遅延ディスパッチ処理を行わないと判断し（ステップＳ４０６、Ｓ４０７）、ステップＳ４１０に進む。最後に、第１の割り込みスレッドは、時刻ｔ８でステップＳ４０２において停止

させた第１のタスク処理スレッドを実行スレッドとして記録し（ステップＳ４１０）、時刻ｔ９でそのスレッドを再開させる（ステップＳ４１１）。これにより、第１のタスク処理スレッドが、再び実行される。このように割り込み処理関数を呼び出した割り込みスレッドが実行中のタスク処理スレッドを停止させ、割り込みハンドラを呼び出した後に、停止させたタスク処理スレッドを再開させることにより、リアルタイムＯＳの割り込み処理を模倣することができる。

【００６６】

図１１は、遅延ディスパッチ処理のタイミングチャートの例である。初期状態から時刻ｔ７までのタイミングチャートは、図１０と一致する。遅延ディスパッチ処理を行う場合、第１の割り込みスレッドは、ステップＳ４０７からステップＳ４０８へ進む。その後、第１の割り込みスレッドは、時刻ｔ８で実行スレッドとして第２のタスク処理スレッドを記録し（ステップＳ４０８）、時刻ｔ９においてそのスレッドを再開させる（ステップＳ４０９）。これにより、時刻ｔ９以降、第１のタスク処理スレッドに代えて、第２のタスク処理スレッドが実行される。このように停止させた割り込みスレッド以外の割り込みスレッドを再開させることにより、リアルタイムＯＳの遅延ディスパッチ処理を模倣することができる。

【００６７】

図１２は、多重割り込み処理のタイミングチャートの例である。なお、図１２（ｂ）は、図１２（ａ）に続く時間帯を示したタイミングチャートである。初期状態から時刻ｔ６－１までのタイミングチャートは、図１０と一致する。第１の割り込みスレッドが時刻ｔ５で割り込み処理を開始可能となった旨を通知した後、第２の割り込みスレッドが時刻ｔ６－１で割り込み処理関数を呼び出すとする。第２の割り込みスレッドは、時刻ｔ６－３で直前に実行されていたスレッド、すなわち、第１の割り込みスレッドを停止させ（ステップＳ４０２）、時刻ｔ６－３からｔ６－７までの期間にステップＳ４０３からＳ４０５までの処理を行う。その後、第２の割り込みスレッドは、多重割り込み処理中であると判断して（ステップＳ４０６）、ステップＳ４０６からステップＳ４１０に進む。最後に、第２の割り込みスレッドは、時刻ｔ６－８で実行スレッドとして第１の割り込み

スレッドを記録し（ステップＳ４１０）し、時刻ｔ６－９でそのスレッドを再開させる（ステップＳ４１１）。時刻ｔ６－９以降、再開された第１の割り込みスレッドは、第２の割り込みスレッドによる割り込みが発生しなかった場合と同様に動作する。このように割り込みスレッド実行中に他の割り込みが発生した時に、先の割り込みスレッドを停止させて、後で発生した割り込みに対応した割り込みハンドラを先に呼び出すことにより、リアルタイムＯＳの多重割り込み処理を模倣することができる。

【００６８】

図１３は、ディスパッチ処理中に割り込みが発生した場合の割り込み処理のタイミングチャートの例である。このタイミングチャートは、図６における時刻ｔ４で、第１の割り込みスレッドが割り込み処理関数を呼び出した場合のものである。第１の割り込みスレッドは、時刻ｔ４－１で第２のイベント待ち状態となる（ステップＳ４０１）。このため、時刻ｔ４－２以降、チェンジャースレッドが実行される。第１の割り込みスレッドは、時刻ｔ７でチェンジャースレッドが第２のイベントを設定した後に実行可能状態となり、時刻ｔ８でチェンジャースレッドが第１のイベント待ち状態となった後に実行される。このようにディスパッチ処理中に割り込みが発生しても、第２のイベントを用いた排他制御を行うことにより、リアルタイムＯＳにおけるディスパッチ処理中の割り込み処理を模倣することができる。

【００６９】

システムクロック割り込みスレッド３３は、システムクロックに対応して擬似的に割り込みを発生させ、システムクロック割り込みに対する割り込みハンドラを備える。このような割り込みスレッドを備えることにより、実システムにおけるリアルタイムＯＳのタイマー管理機能を模倣することができる。

【００７０】

以上に示すように、本実施形態に係るリアルタイムＯＳシミュレータでは、タスク処理スレッドは、システム関数を呼び出した時に、次に実行すべきタスク処理スレッドを選択し、チェンジャータスクに対してディスパッチ処理開始を指示した後に、自ら停止する。一方、チェンジャースレッドは、処理開始を指示され

た時には、直前に実行されていたタスク処理スレッドが停止した後に、次に処理すべく選択されたタスク処理スレッドを再開させる。これにより、汎用ＯＳが提供するスレッドスケジューリングアルゴリズムにかかわらず、リアルタイムＯＳのディスパッチ処理を模倣することができる。また、タスク処理スレッドは、チェンジャースレッドがディスパッチ処理を可能状態になった後に処理開始を指示するので、複数のタスク処理スレッドが同時にディスパッチ処理を行うことがない。さらに、割り込みスレッドは、割り込み処理関数を呼び出した時に、実行中のスレッドを停止させて、対応した割り込みハンドラを呼び出した後に、次に実行すべきタスク処理スレッドを再開させる。これにより、リアルタイムＯＳの割り込み処理と遅延ディスパッチ処理とを模倣することができる。また、割り込みスレッドは、割り込みスレッド実行中に割り込みが発生した時にも、実行中の割り込みスレッドを停止させ、後の割り込みに対応した割り込みハンドラを呼び出した後に、停止させた割り込みスレッドを再開させる。これにより、リアルタイムＯＳにおける多重割り込み処理を模倣することができる。このように、本実施形態に係るリアルタイムＯＳシミュレータは、リアルタイムＯＳ上で実行されるマルチタスクのソフトウェアを開発するために必要とされるリアルタイムＯＳの機能を模倣することができる。

【００７１】

（第２の実施形態）

本発明の第２の実施形態に係るリアルタイムＯＳシミュレータは、第１の実施形態に係るリアルタイムＯＳシミュレータと同一のソフトウェア構成を有し、チェンジャースレッドおよびシステム関数の一部のみが異なる。本実施形態に係るリアルタイムＯＳシミュレータは、自スレッドを停止させることを禁止する汎用ＯＳ上でも動作する。このリアルタイムＯＳシミュレータでは、各タスク処理スレッドごとに設けられ、各タスク処理スレッドが待機状態にあることを示す第３のイベントが使用される。

【００７２】

図１４は、本実施形態に係るチェンジャースレッド１１の動作を示すフローチャートである。第１の実施形態に係るチェンジャースレッドは、図４に示すよう

に、実行中のタスク処理スレッドが停止するまで、所定の時間だけスリープする処理を繰り返す（ステップＳ２０３、Ｓ２０４）。これに対し、本実施形態に係るチェンジャースレッドは、実行中のタスク処理スレッドを停止させる（ステップＳ５０３）ことを特徴とする。

【００７３】

図１４に示すフローチャートにおいて、ステップＳ５０３からＳ５０５まで以外の処理は図４と同一であるので、説明を省略する。チェンジャースレッドは、ディスパッチ処理開始を指示する第１のイベントが設定された（ステップＳ５０２）後に、実行中のタスク処理スレッドを停止させる（ステップＳ５０３）。次に、チェンジャースレッドは、次に実行すべきタスク処理スレッドが割り込み時の遅延ディスパッチ処理によって停止しているか否かを判断し（ステップＳ５０４）、遅延ディスパッチによって停止していると判断した場合には、次に実行すべきタスク処理スレッドに係る第３のイベントを設定する（ステップＳ５０５）。

【００７４】

図１５は、タスク処理スレッドから呼び出されるシステム関数１２のフローチャートである。このフローチャートにおけるステップＳ６０１からＳ６０４の処理は、図５に示すフローチャートにおけるステップＳ４０１からＳ４０４と同一であるので、説明を省略する。システム関数を呼び出したタスク処理スレッドは、ステップＳ６０５において、自タスク処理スレッドに係る第３のイベント待ち状態となる。

【００７５】

このように、タスク処理スレッドはチェンジャースレッドに対して処理開始を指示した後に第３のイベント待ち状態となり、チェンジャースレッドは処理開始を指示されると、実行中のタスク処理スレッドを停止させる。このため、自スレッドを停止させることを禁止する汎用のマルチスレッドＯＳを用いた場合でも、リアルタイムＯＳのディスパッチ処理を模倣することができる。

【００７６】

図１６および図１７は、本実施形態に係るリアルタイムＯＳシミュレータによ

るディスパッチ処理のタイミングチャートの例である。図１６および図１７は、いずれも図６と同じ記法を用いて記述され、それぞれ図６および図７に対応する。初期状態では、第２のイベントは設定されているとする。また、チェンジャースレッドは第１のイベント待ち状態（ステップＳ５０２）にあり、第２のタスク処理スレッドはシステム関数の中で第３のイベント待ち状態（ステップＳ６０５）にあるとする。

【００７７】

図１６に示すタイミングチャートでは、時刻ｔ１でシステム関数を呼び出した第１のタスク処理スレッドは、時刻ｔ３で第１のイベントを設定（ステップＳ６０４）した後、時刻ｔ３－１で第３のイベント待ち状態となる（ステップＳ６０５）。このため、時刻ｔ３－１以降、チェンジャースレッドが実行される。チェンジャースレッドは、時刻ｔ４で第１のタスク処理スレッドを停止させる（ステップＳ５０３）。チェンジャースレッドは、次に実行すべきタスク処理スレッドが遅延ディスパッチ処理によって停止しているか否かを判断する（ステップＳ５０４）。この例では、チェンジャースレッドは、ステップＳ５０４からステップＳ５０５へ進み、時刻ｔ４－１で第２のタスク処理スレッドに係る第３のイベントを設定する（ステップＳ５０５）。このため、時刻ｔ４－１以降、第２のタスク処理スレッドは、第３のイベント待ち状態から抜けて停止する。その後、チェンジャースレッドは、時刻ｔ５で次に実行すべきスレッドとして第２のタスク処理スレッドを記録した後（ステップＳ５０６）、時刻ｔ６で第２のタスク処理スレッドを再開させる（ステップＳ５０７）。このため、時刻ｔ６以降、第２のタスク処理スレッドは、実行可能状態となる。さらに、チェンジャースレッドは、時刻ｔ７で第２のイベントを設定した後（ステップＳ５０８）、第１のイベント待ち状態となる（ステップＳ５０２）。このため、時刻ｔ８以降、第２のタスク処理スレッドが実行される。

【００７８】

図１７は、時刻ｔ３以降にチェンジャースレッドが実行される場合のタイミングチャートの例である。このタイミングチャートは、チェンジャータスクが時刻ｔ４で第１のタスク処理スレッドを停止させる（ステップＳ５０３）点でのみ、

図１６と異なる。

【００７９】

なお、図１６および図１７のタイミングチャートにおいて、第２のタスク処理スレッドが第３のイベント待ち状態でない場合には、チェンジャースレッドは、ステップＳ５０４からステップＳ５０６に進み、第２のタスク処理スレッドに係る第３のイベントを設定しない。

【００８０】

以上に示すように、本実施形態に係るリアルタイムＯＳシミュレータでは、タスク処理スレッドは、チェンジャースレッドに対して処理開始を指示した後に待機状態となる。チェンジャースレッドは、処理開始を指示されると実行中のタスク処理スレッドを停止させた後に、次に実行すべきタスク処理スレッドが待機状態にある場合は、その待機状態を解除して再開させる。このため、自スレッドを停止させることを禁止する汎用のマルチスレッドＯＳを用いた場合でも、リアルタイムＯＳのディスパッチ処理を模倣することができる。

【００８１】

なお、指定された優先度に従ってスレッドを実行させる汎用のマルチスレッドＯＳを用いる場合には、スケジューラスレッドの優先度をタスク処理スレッドよりも高く設定する方法を用いてもよい。この方法によれば、図１４および図１５に示すフローチャートから第３のイベントに関する処理を削除しても、スケジューラスレッドがタスク処理スレッドよりも優先的に実行され、本実施形態の効果が損なわれることがない。

【００８２】

以上に示した第１および第２の実施形態に係るリアルタイムＯＳシミュレータについては、リアルタイムＯＳおよび汎用ＯＳは、特定のＯＳに限られるものではない。また、リアルタイムＯＳシミュレータは、汎用ＯＳが提供するイベント機能以外の機能を用いて、スレッド間の同期を実現するものであってもよい。また、タスク処理スレッドおよび割り込みスレッドは、いずれも、リアルタイムＯＳシミュレータにより生成されても、リアルタイムＯＳシミュレータの外部で生成されてもよい。

【図面の簡単な説明】

【図１】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータが動作するコンピュータシステムのハードウェア構成図である。

【図２】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータのソフトウェア構成図である。

【図３】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータのメイン処理のフローチャートである。

【図４】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータにおけるチェンジャースレッドのフローチャートである。

【図５】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータにおけるシステム関数のフローチャートである。

【図６】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータによるディスパッチ処理のタイミングチャートの例である。

【図７】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータによるディスパッチ処理のタイミングチャートの例である。

【図８】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータによるディスパッチ処理のタイミングチャートの例である。

【図９】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータの割り込み処理関数のフローチャートである。

【図１０】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータによる割り込み処理のタイミングチャートの例である。

【図１１】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータによる割り込み処理のタイミングチャートの例である。

【図１２】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータによる割り込み処理のタイミングチャートの例である。

【図１３】

本発明の第１の実施形態に係るリアルタイムＯＳシミュレータによる割り込み処理のタイミングチャートの例である。

【図１４】

本発明の第２の実施形態に係るリアルタイムＯＳシミュレータにおけるチェンジャースレッドのフローチャートである。

【図１５】

本発明の第２の実施形態に係るリアルタイムＯＳシミュレータにおけるシステム関数のフローチャートである。

【図１６】

本発明の第２の実施形態に係るリアルタイムＯＳシミュレータによるディスパッチ処理のタイミングチャートの例である。

【図１７】

本発明の第２の実施形態に係るリアルタイムＯＳシミュレータによるディスパッ処理のタイミングチャートの例である。

【図１８】

従来の制御ソフトウェア実行システムのソフトウェア構成図である。

【符号の説明】

１０…リアルタイムＯＳシミュレータ

１１…チェンジャースレッド

１２…システム関数

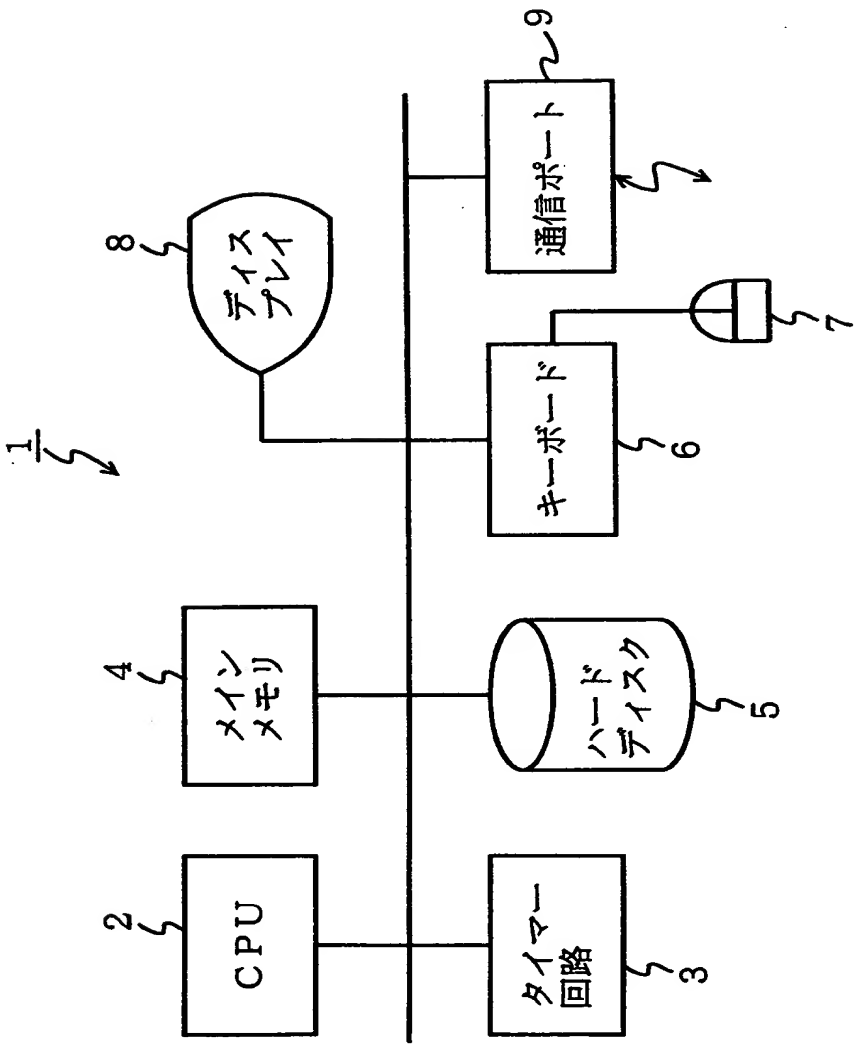
１３…割り込み処理関数

２１～２３…タスク処理スレッド

３１～３３…割り込みスレッド

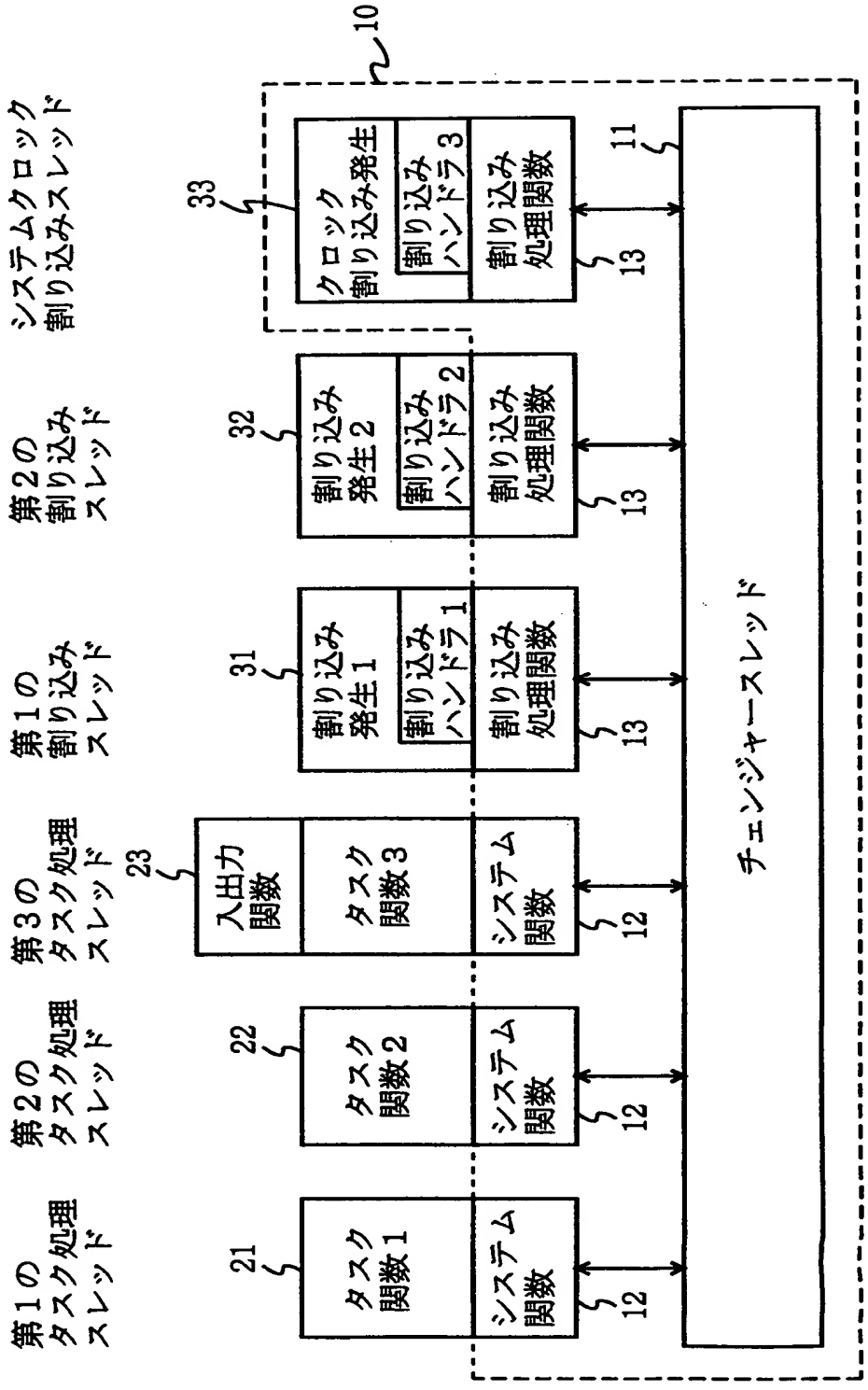
【書類名】　　図面

【図１】

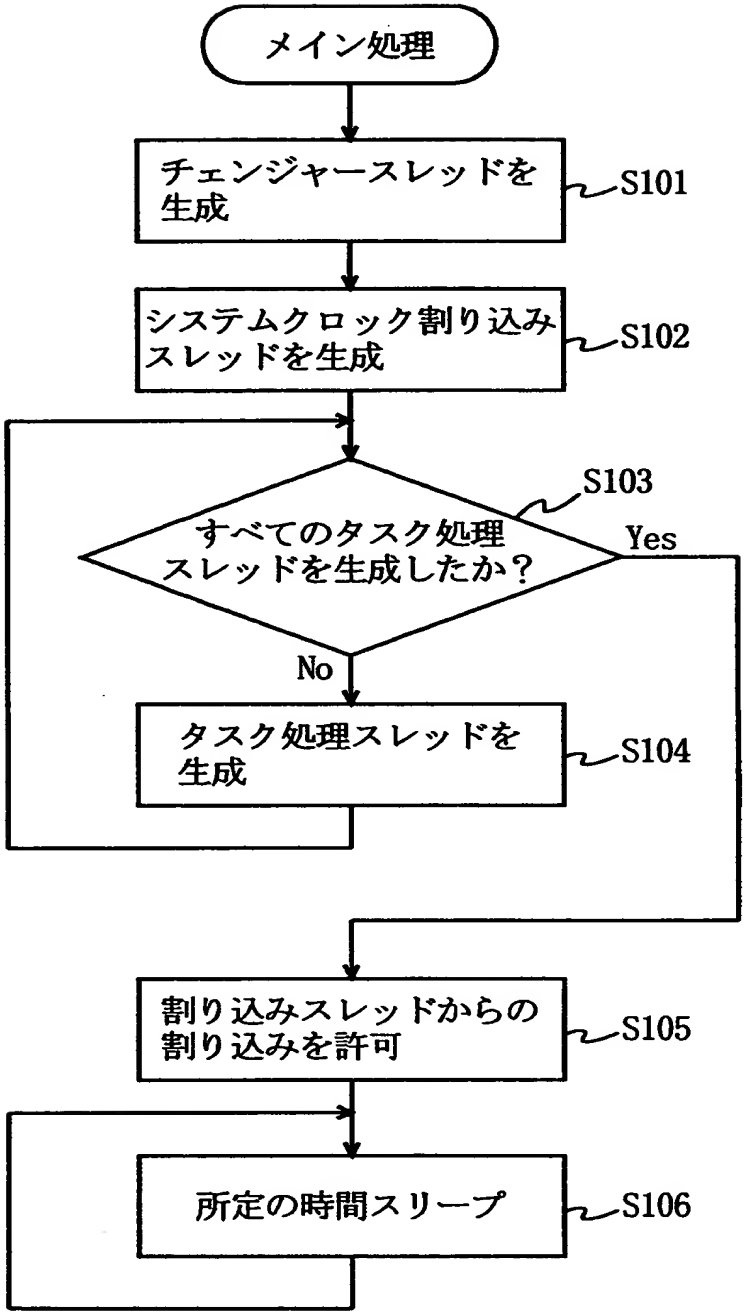


出証特２０００－３１０４１５８

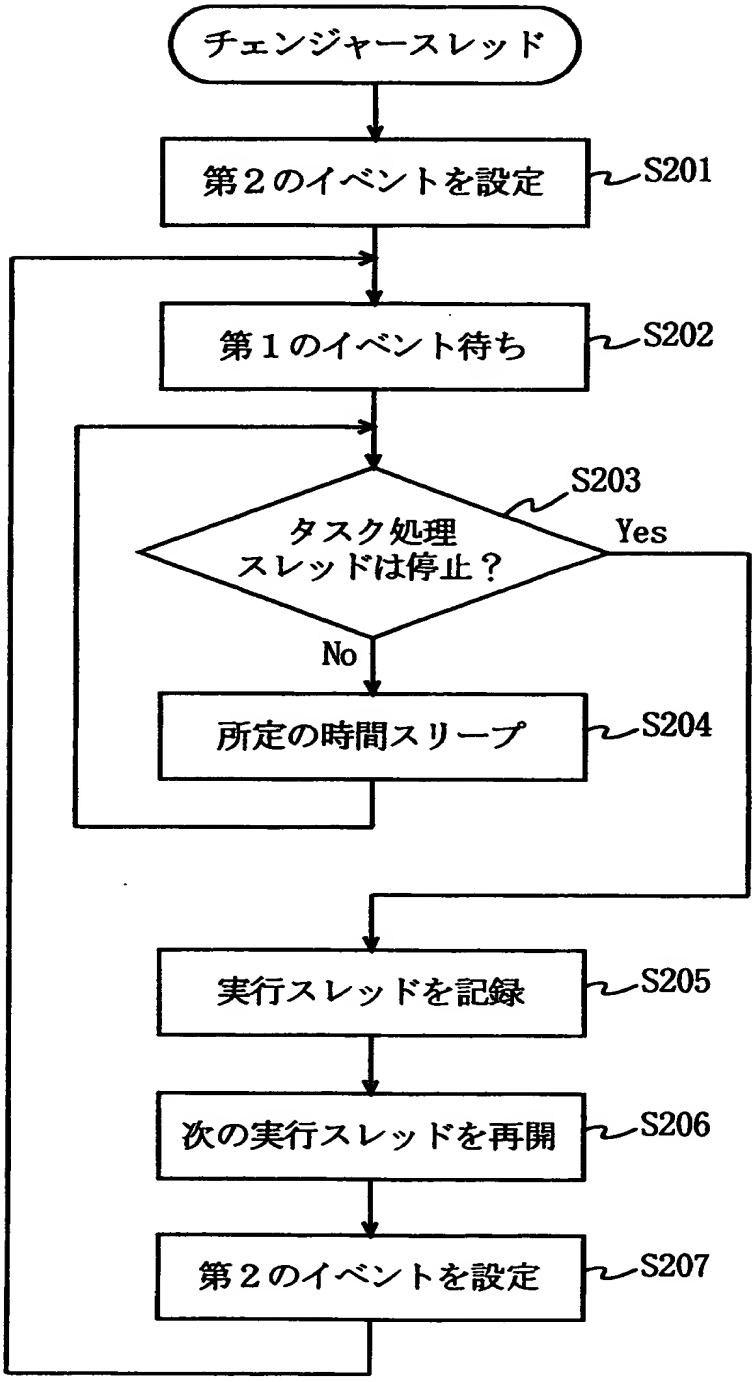
【図２】


第１の
タスク処理
スレッド
第２の
タスク処理
スレッド
第３の
タスク処理
スレッド
第１の
割り込み
スレッド
第２の
割り込み
スレッド
システムクロック
割り込みスレッド
10
21
タスク
関数１
システム
関数
12
22
タスク
関数２
システム
関数
12
23
入出力
関数
タスク
関数３
システム
関数
12
31
割り込み
発生１
割り込み
ハンドラ１
割り込み
処理関数
13
32
割り込み
発生２
割り込み
ハンドラ２
割り込み
処理関数
13
33
クロック
割り込み発生
割り込み
ハンドラ３
割り込み
処理関数
13
11
チェンジャースレッド

【図３】

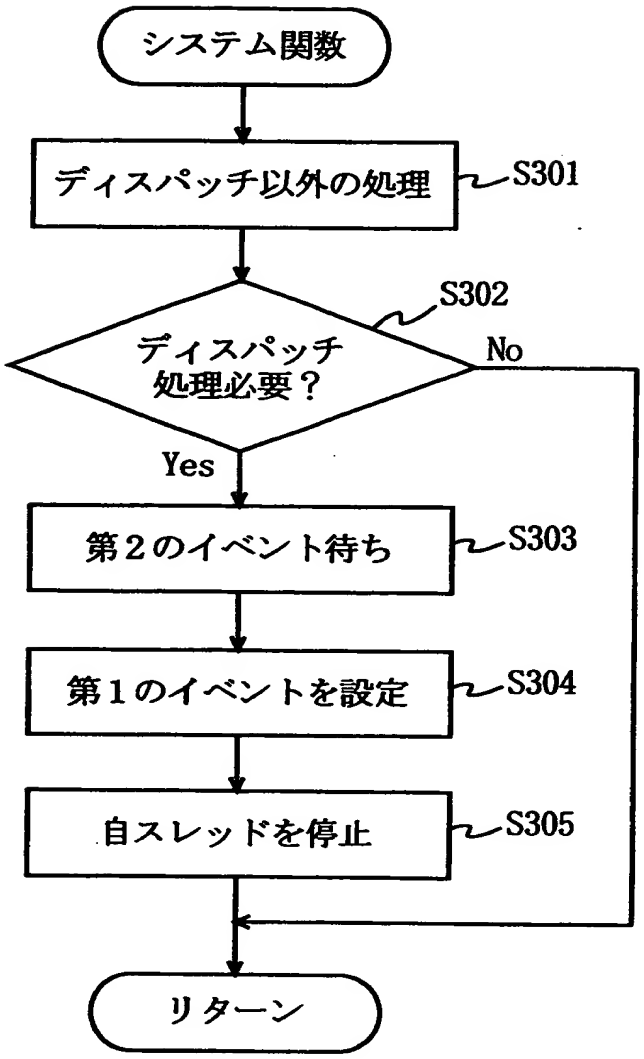

メイン処理
チェンジャースレッドを生成
S101
システムクロック割り込みスレッドを生成
S102
S103
すべてのタスク処理スレッドを生成したか？
Yes
No
タスク処理スレッドを生成
S104
割り込みスレッドからの割り込みを許可
S105
所定の時間スリープ
S106

【図４】

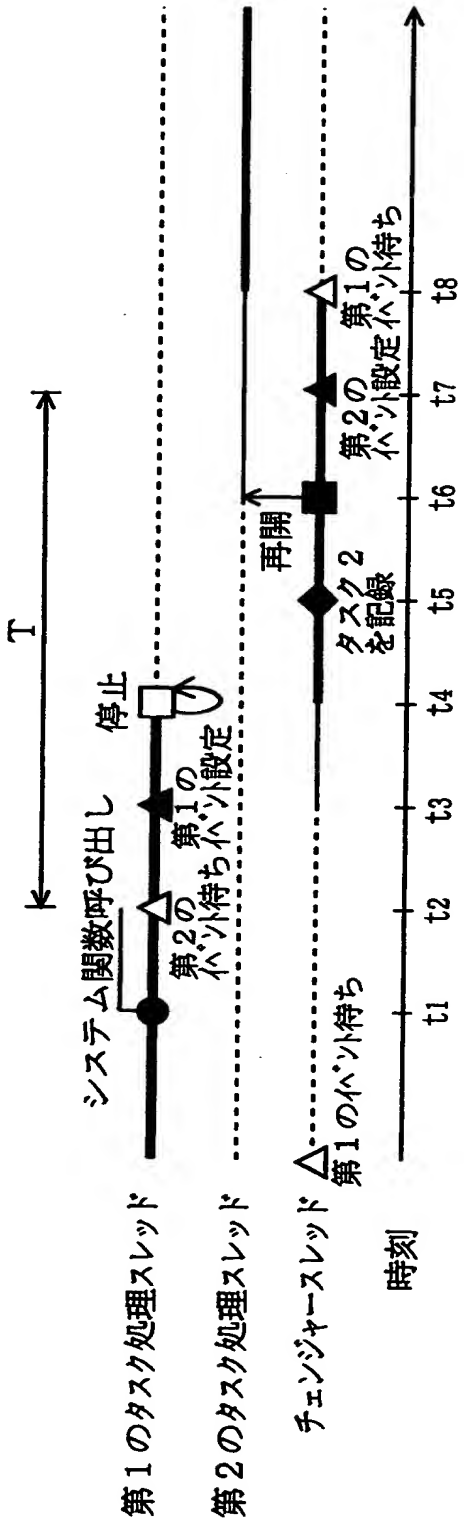


出証特２０００－３１０４１５８

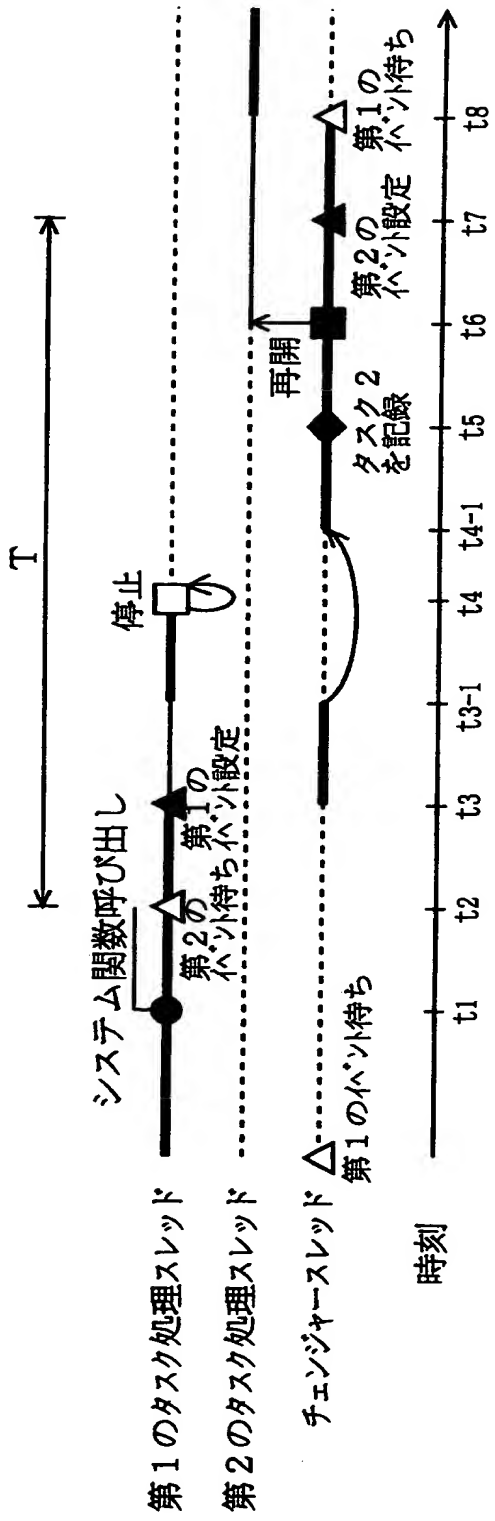
【図５】


システム関数
ディスパッチ以外の処理
S301
S302
ディスパッチ
処理必要？
No
Yes
第２のイベント待ち
S303
第１のイベントを設定
S304
自スレッドを停止
S305
リターン

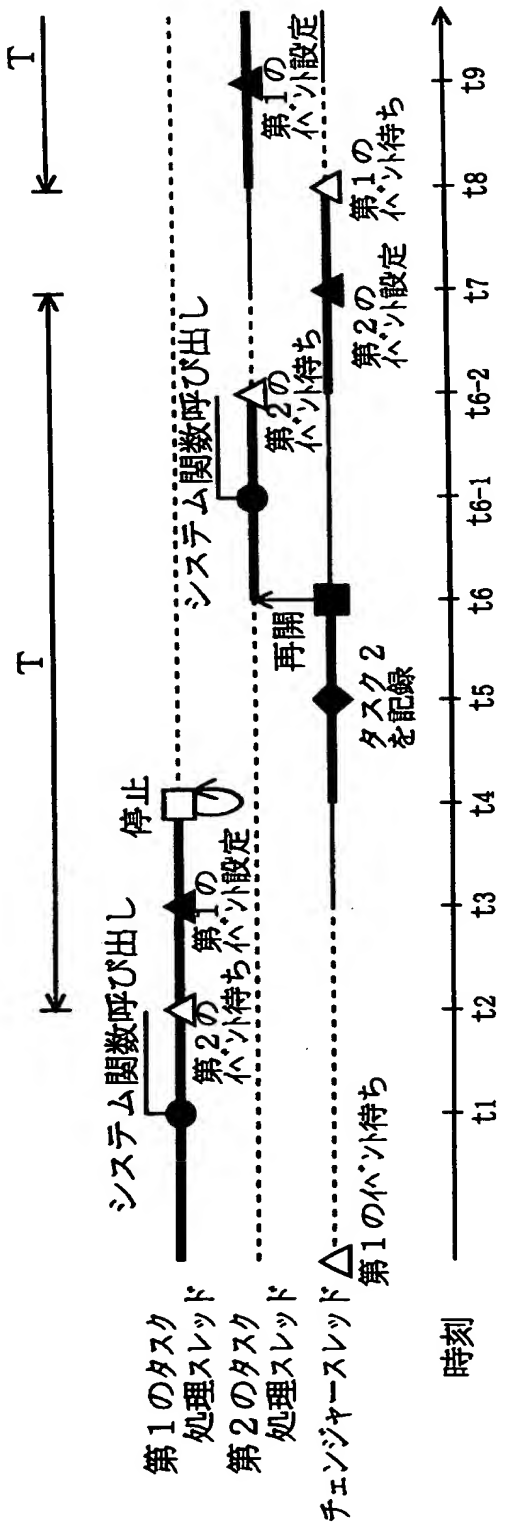
【図６】


T
システム関数呼び出し
停止
第１のタスク処理スレッド
第２のイベント待ち
第１のイベント設定
第２のタスク処理スレッド
再開
チェンジャースレッド
第１のイベント待ち
タスク２を記録
第２のイベント設定
第１のイベント待ち
t1
t2
t3
t4
t5
t6
t7
t8
時刻

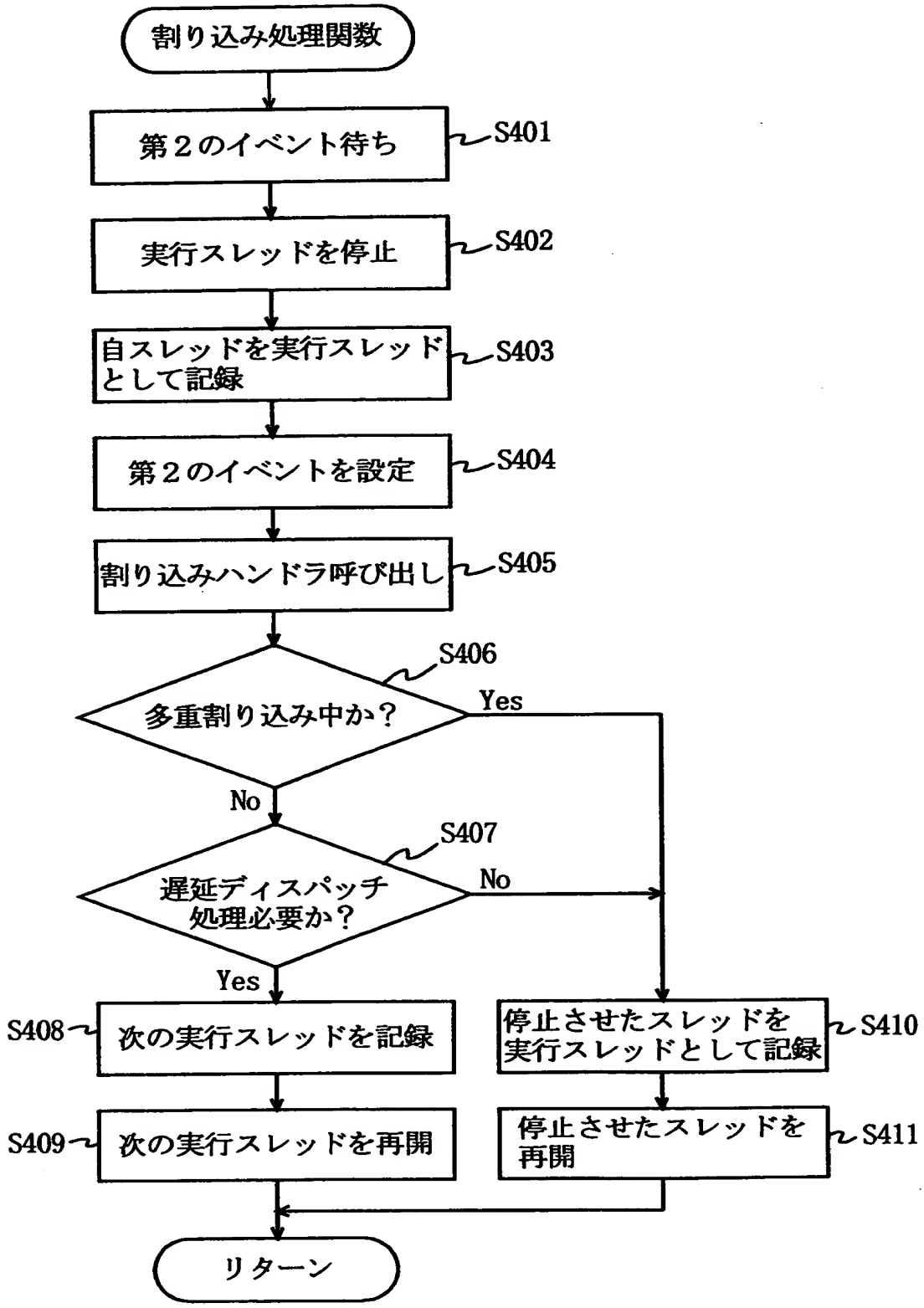
【図７】


T
システム関数呼び出し
停止
第２の
イベント待ち
第１の
イベント設定
再開
タスク２
を記録
第２の
イベント設定
第１の
イベント待ち
第１のタスク処理スレッド
第２のタスク処理スレッド
チェンジャースレッド
第１のイベント待ち
時刻
t1
t2
t3
t3-1
t4
t4-1
t5
t6
t7
t8

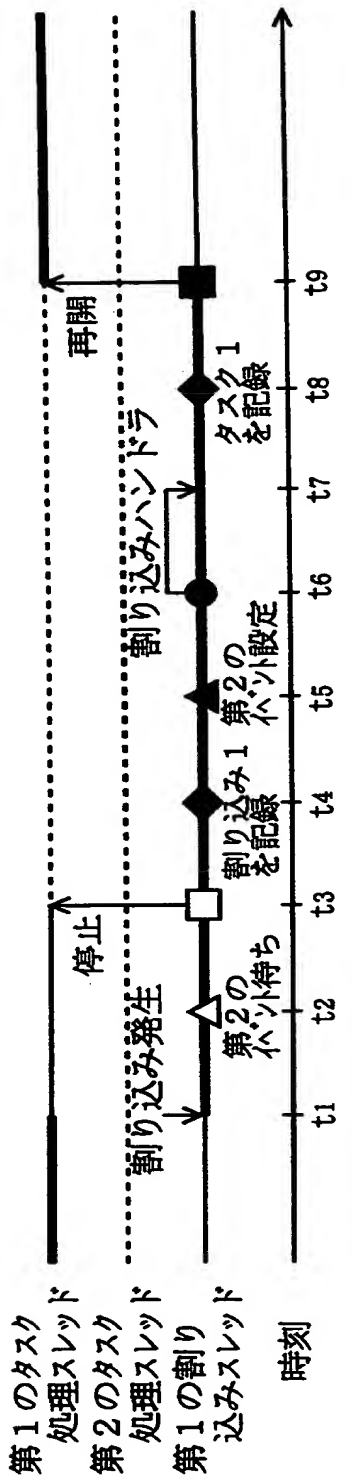
【図８】


T
T
システム関数呼び出し
停止
第２のイベント待ち
第１のイベント設定
システム関数呼び出し
第２のイベント待ち
再開
タスク２を記録
第２のイベント設定
第１のイベント待ち
第１のイベント設定
第１のイベント待ち
第１のタスク処理スレッド
第２のタスク処理スレッド
チェンジャースレッド
時刻
t1
t2
t3
t4
t5
t6
t6-1
t6-2
t7
t8
t9

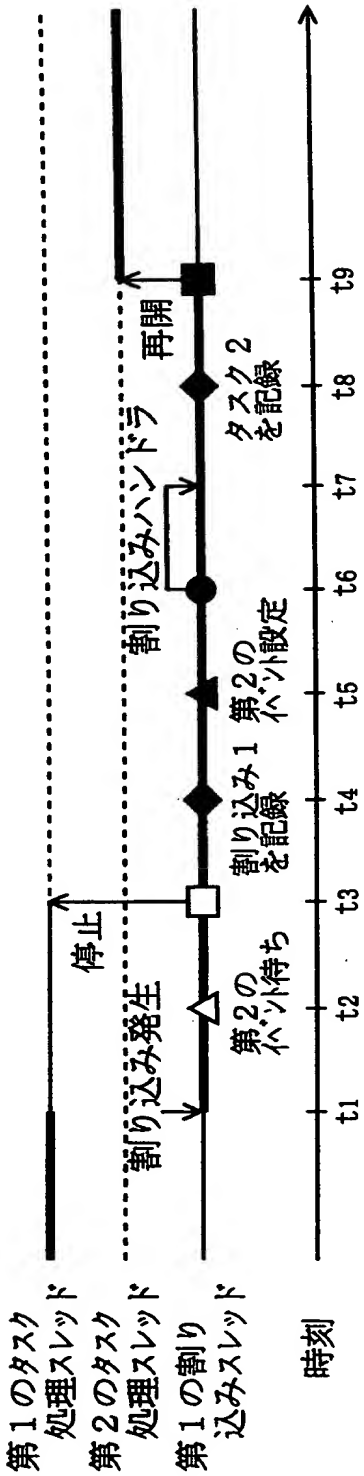
【図９】


割り込み処理関数
第２のイベント待ち
S401
実行スレッドを停止
S402
自スレッドを実行スレッドとして記録
S403
第２のイベントを設定
S404
割り込みハンドラ呼び出し
S405
S406
多重割り込み中か？
Yes
No
S407
遅延ディスパッチ処理必要か？
No
Yes
S408
次の実行スレッドを記録
S409
次の実行スレッドを再開
停止させたスレッドを実行スレッドとして記録
S410
停止させたスレッドを再開
S411
リターン

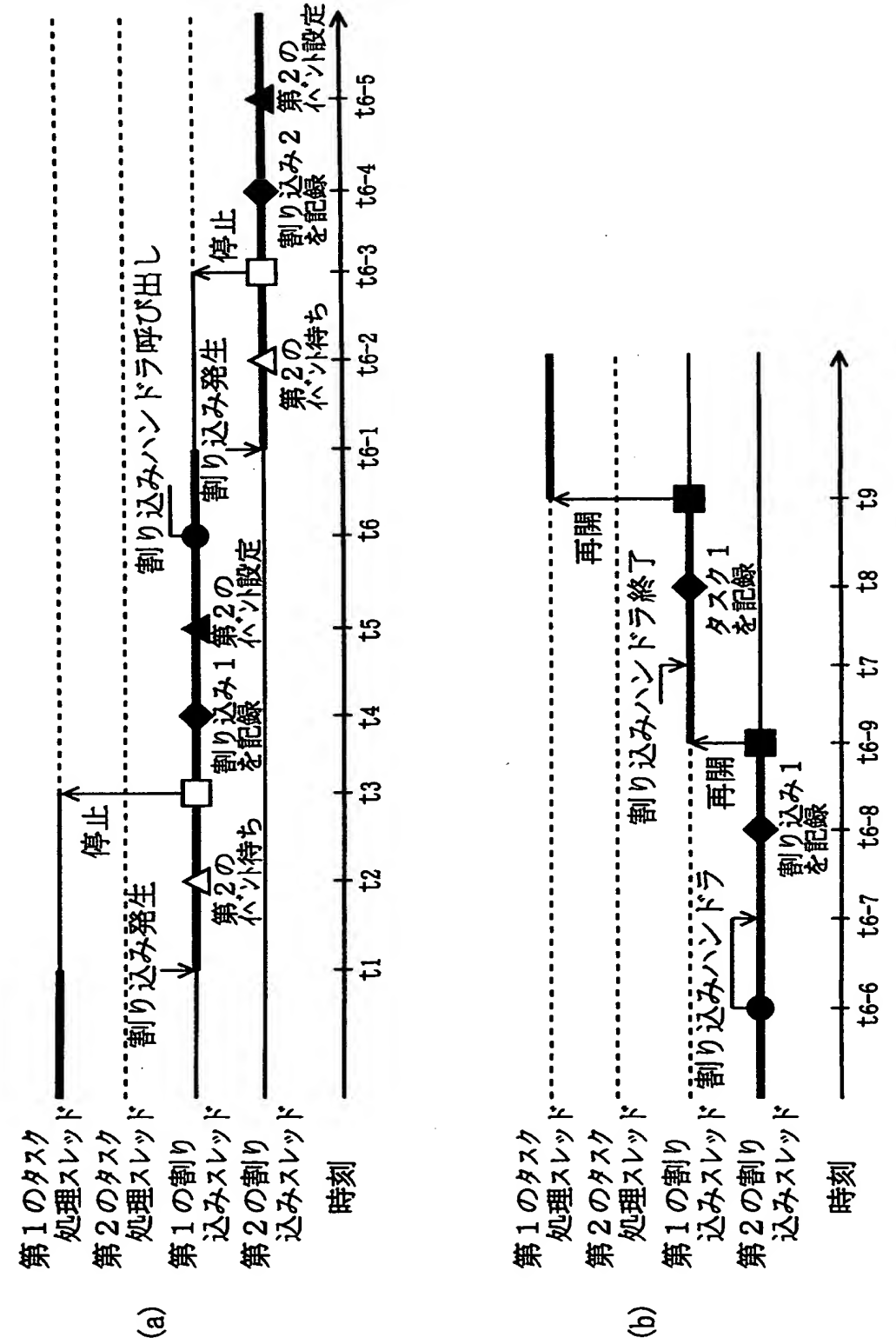
【図１０】


第１のタスク処理スレッド
第２のタスク処理スレッド
第１の割り込みスレッド
時刻
t1
t2
t3
t4
t5
t6
t7
t8
t9
割り込み発生
停止
第２のイベント待ち
割り込み１を記録
第２のイベント設定
割り込みハンドラ
タスク１を記録
再開

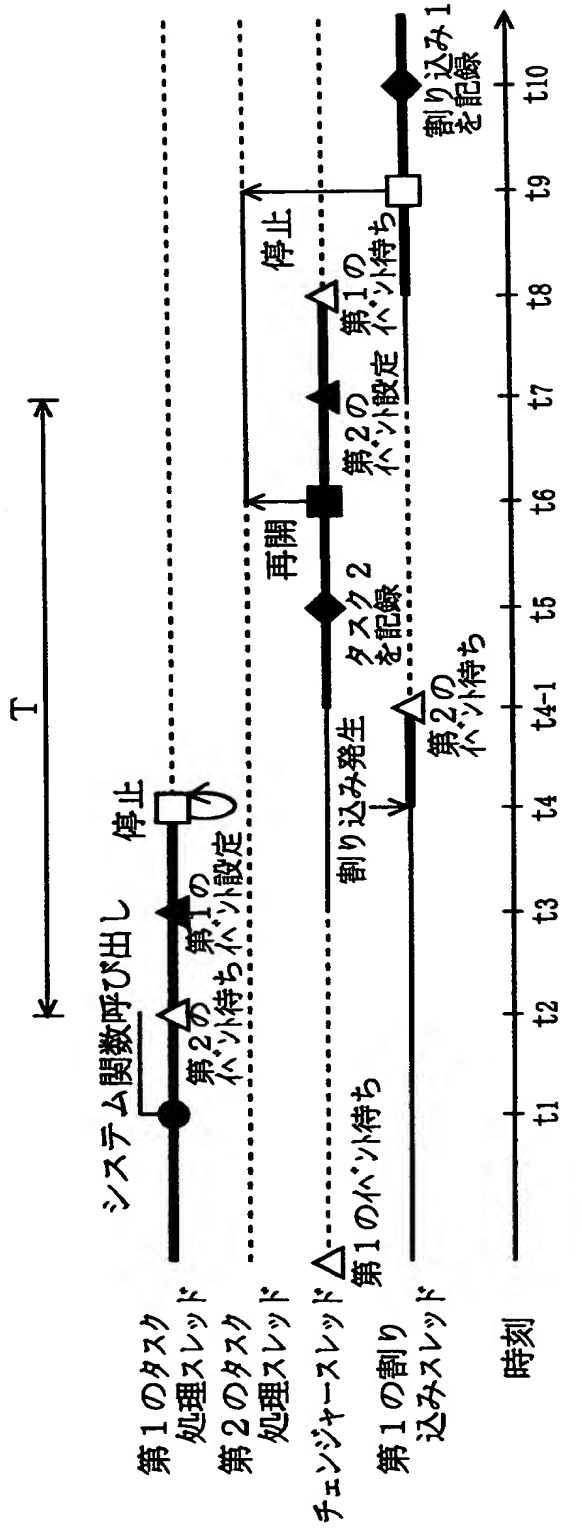
【図１１】


第１のタスク処理スレッド
第２のタスク処理スレッド
第１の割り込みスレッド
時刻
t1
t2
t3
t4
t5
t6
t7
t8
t9
停止
割り込み発生
第２のイベント待ち
割り込み１を記録
第２のイベント設定
割り込みハンドラ
再開
タスク２を記録

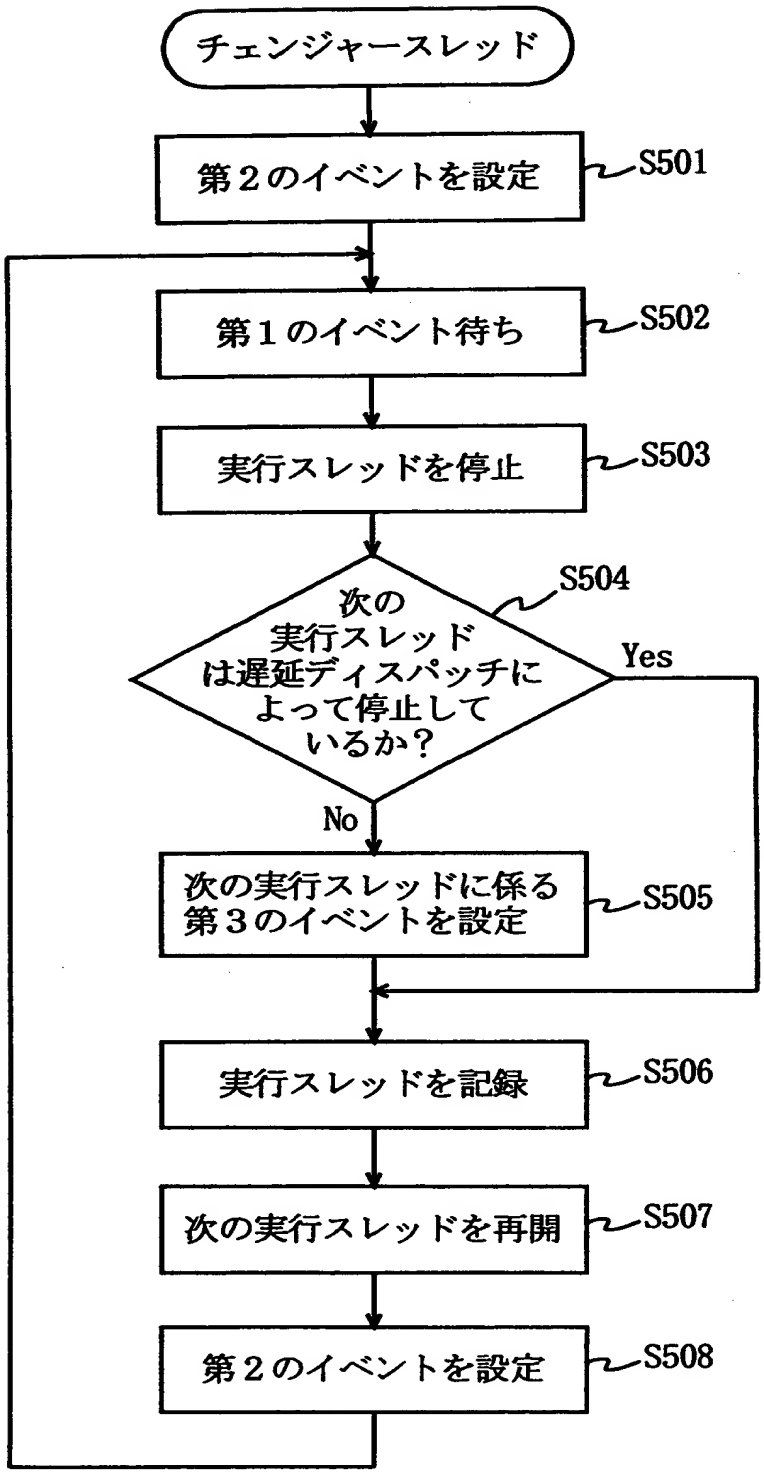
【図１２】


(a)
第１のタスク処理スレッド
第２のタスク処理スレッド
第１の割り込みスレッド
第２の割り込みスレッド
時刻
停止
割り込みハンドラ呼び出し
割り込み発生
第２のポント待ち
割り込み１を記録
第２のポント設定
停止
割り込み発生
第２のポント待ち
割り込み２を記録
第２のポント設定
t1
t2
t3
t4
t5
t6
t6-1
t6-2
t6-3
t6-4
t6-5
(b)
第１のタスク処理スレッド
第２のタスク処理スレッド
第１の割り込みスレッド
第２の割り込みスレッド
時刻
再開
割り込みハンドラ終了
タスク１を記録
再開
割り込みハンドラ
割り込み１を記録
t6-6
t6-7
t6-8
t6-9
t7
t8
t9

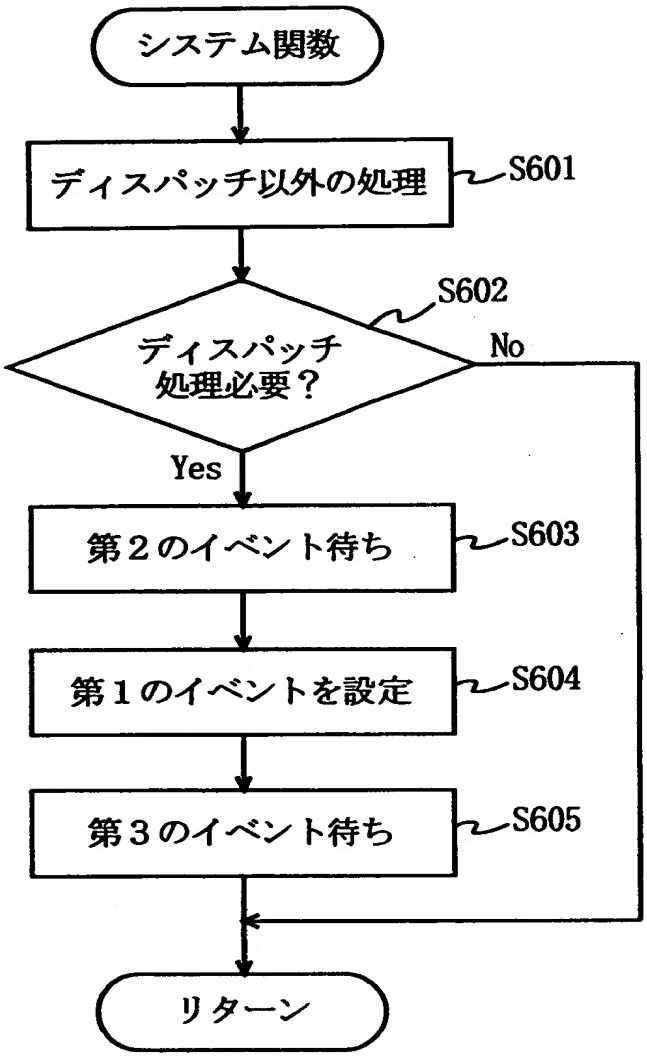
【図１３】


T
システム関数呼び出し
停止
第２のイベント待ち
第１のイベント設定
再開
第２のイベント設定
第１のイベント待ち
割り込みを記録
第１のタスク処理スレッド
第２のタスク処理スレッド
チェンジャースレッド
第１の割り込みスレッド
第１のイベント待ち
割り込み発生
タスク２を記録
第２のイベント待ち
時刻
t1
t2
t3
t4
t4-1
t5
t6
t7
t8
t9
t10

【図１４】

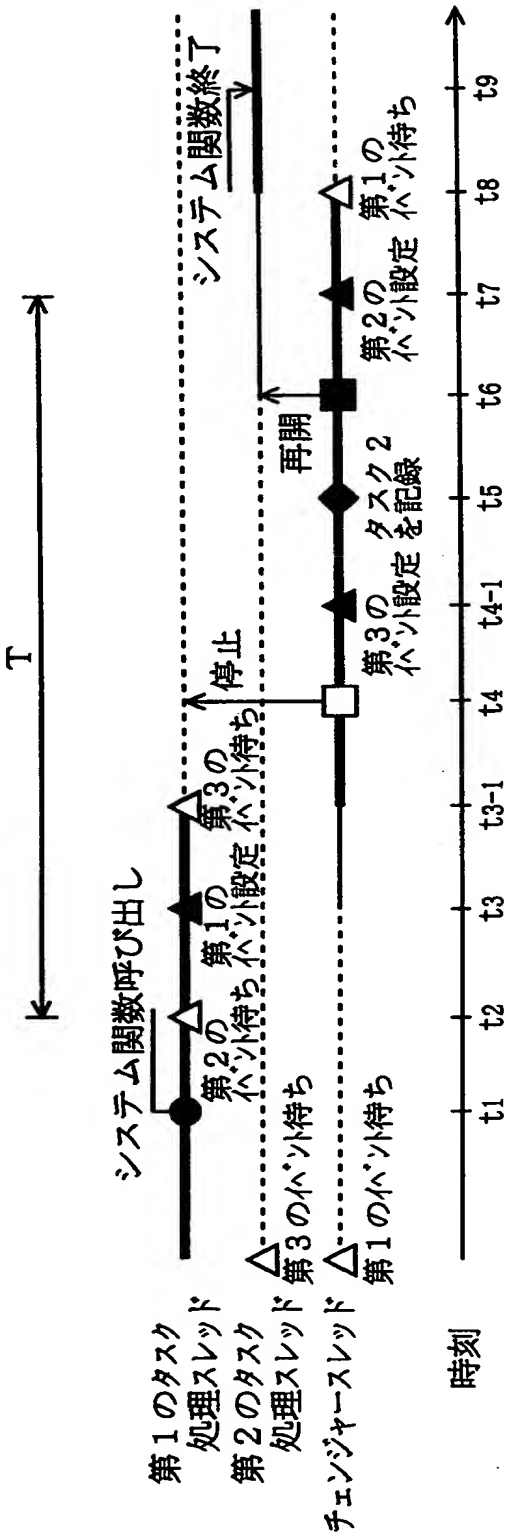

チェンジャースレッド
第２のイベントを設定
S501
第１のイベント待ち
S502
実行スレッドを停止
S503
次の
実行スレッド
は遅延ディスパッチに
よって停止して
いるか？
S504
Yes
No
次の実行スレッドに係る
第３のイベントを設定
S505
実行スレッドを記録
S506
次の実行スレッドを再開
S507
第２のイベントを設定
S508

【図１５】

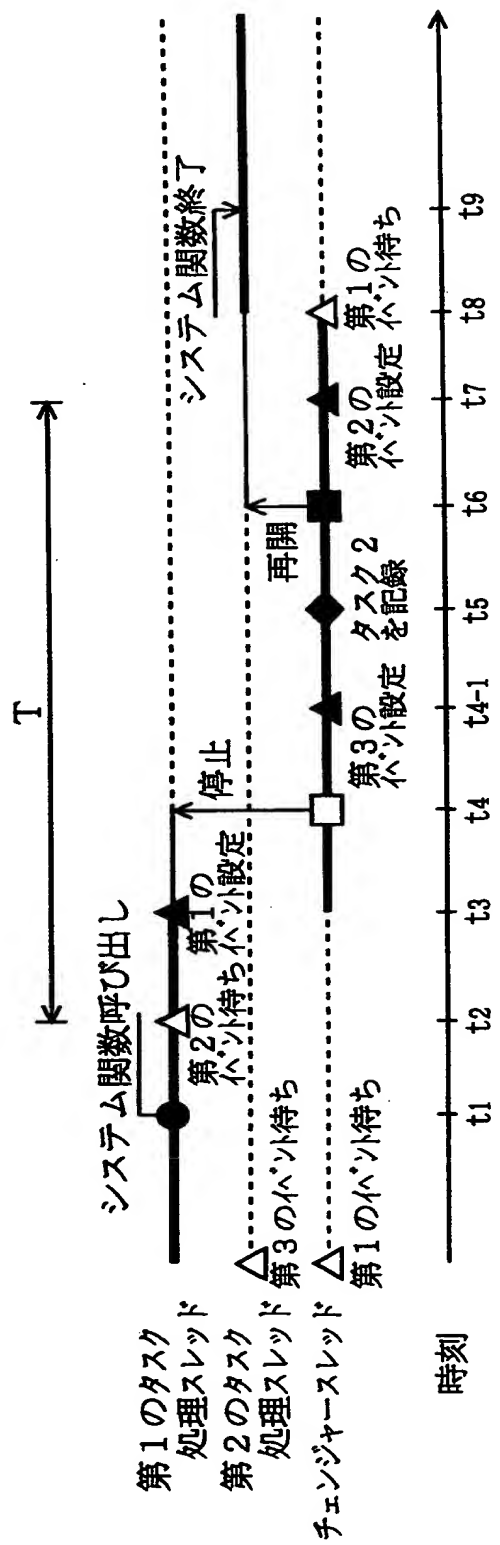


出証特２０００－３１０４１５８

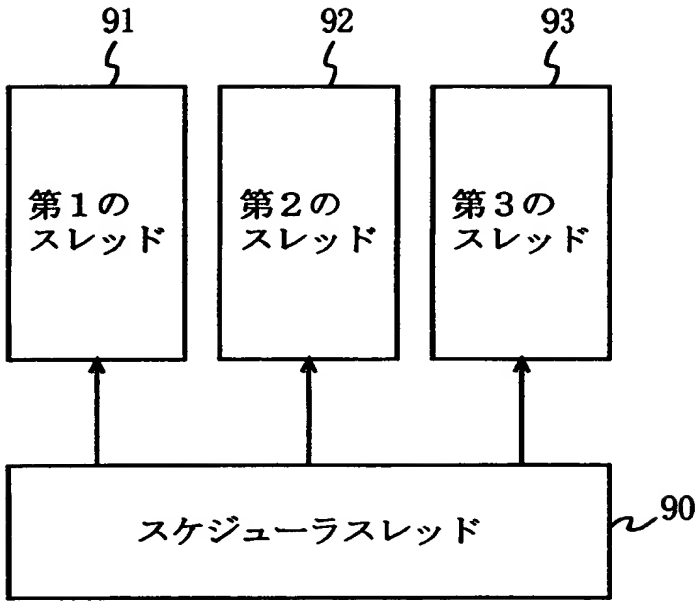
【図１６】


T
システム関数呼び出し
システム関数終了
第１のタスク
処理スレッド
第２のタスク
処理スレッド
チェンジャースレッド
第２の
イベント待ち
第１の
イベント設定
第３の
イベント待ち
停止
再開
第３のイベント待ち
第１のイベント待ち
第３の
イベント設定
タスク２
を記録
第２の
イベント設定
第１の
イベント待ち
時刻
t1
t2
t3
t3-1
t4
t4-1
t5
t6
t7
t8
t9

【図１７】


T
システム関数呼び出し
第２のイベント待ち
第１のイベント設定
停止
システム関数終了
再開
第３のイベント設定
タスク２を記録
第２のイベント設定
第１のイベント待ち
第１のタスク処理スレッド
第２のタスク処理スレッド
チェンジャースレッド
第３のイベント待ち
第１のイベント待ち
t1
t2
t3
t4
t4-1
t5
t6
t7
t8
t9
時刻

【図１８】



【書類名】　　要約書

【要約】

【課題】　リアルタイムＯＳのディスパッチ処理と割り込み処理とを、汎用のマルチスレッドＯＳ上で模倣する。

【解決手段】　タスクに対応したスレッド２１～２３、擬似的な割り込みを発生させるスレッド３１～３３、スレッドの実行を制御するスレッド１１が、マルチスレッドＯＳ上で並行に実行される。システム関数１２を呼び出したスレッド２１～２３は、次に実行すべきスレッドを選択して、スレッド１１に処理開始を指示した後、自ら停止する。指示を受けたスレッド１１は、実行中のスレッドが停止した後に、選択されたスレッドを再開させる。割り込み処理関数１３を呼び出したスレッド３１～３３は、実行中のスレッドを停止させ、割り込みハンドラを呼び出し後に、次に実行すべきスレッドを選択して再開させる。

【選択図】　　図２

認定・付加情報

特許出願の番号	特願２０００－１２０９０３
受付番号	５００００５０７６６７
書類名	特許願
担当官	第七担当上席　　００９６
作成日	平成１２年　４月２４日

＜認定情報・付加情報＞

【提出日】　平成12年 4月21日

次頁無

出　願　人　履　歴　情　報

識別番号　　〔０００００５８２１〕

１．変更年月日　　１９９０年　８月２８日
［変更理由］　新規登録
住　所　　大阪府門真市大字門真１００６番地
氏　名　　松下電器産業株式会社